CONFLICT CAUSE IDENTIFICATION IN WEB-BASED CONCURRENT ENGINEERING DESIGN

By

TIANHONG JIANG

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2000

ACKNOWLEDGEMENTS

I am grateful to the chairman of my committee, Dr. Gale E. Nevill, Jr., who introduced this area of research to me. I would also like to thank him for his guidance, both professionally and personally, and also for his tremendous encouragement, advice and support during my doctoral work.

I would also like to thank the other members in my committee, Dr. Raphael T. Haftka, Dr. Andrew J. Kurdila, Dr. Paul Fishwick, and Dr. Joseph Duffy for their valued advice, expertise, and support which helped in the development of this research project.

I would also like to thank Dr. Chang-Tsan Sun for his support and guidance during my first two years in this department and his continued friendship.

I would also like to thank Dr. Douglas D. Dankel and Dr. Bhavani V. Sankar for their valued advice in this research project.

I thank my colleague Ruiqiang Zhuang for his cooperation and support in this project. I would also like to thank my fellow graduate students Yunfei Feng, Weijian Wang, Dazhi Yu for their support in the experiments in this project.

Last, but not least, I would like to thank my wife, Yiting Li, for her love, support and understanding during the most trying years of my life and to my family in Pennsylvania, Minnesota, and China for their love and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

CONFLICT CAUSE IDENTIFICATION IN WEB-BASED CONCURRENT ENGINEERING DESIGN

By

Tianhong Jiang

May 2000

Chairman: Dr. Gale E. Nevill, Jr.
Major Department: Aerospace Engineering, Mechanics and Engineering Science

This dissertation explores models and methods for Conflict Cause Identification (C^2I) in web-based concurrent engineering (CE) design. Conflict classes and causes in CE design are identified, defined and cataloged. C^2I methods are studied, organized and tested. The test-bed for this work is the Plane World aircraft design simulation program, created to support exploration of Conflict Cause Identification in a distributed, multi-agent CE design environment.

CHAPTER 1
INTRODUCTION

1.1 Traditional Design vs. Concurrent Design

Engineering design is a highly complex activity. Traditionally, design activity is divided into a sequence of sub-tasks and allocated to multiple designers or sub-contractors (Harrington 1995). For example, the life cycle of a product can be divided into three stages: product design, manufacture, and operation/maintenance. The traditional design approach handles these stages in sequence (Figure 1-1). In each stage, different design criteria are applied; i.e. performance is the critical factor considered in the product design phase while manufactureability and cost are the main concerns in manufacturing.

While relatively simple to manage, the traditional approach has some serious drawbacks (Harrington 1996). Designers involved in various stages make their decisions based on their local goals with little concern for the other stages. This commonly causes conflicts[1] of design between the early and later stages and results in sub-optimal results. Re-work and Compromise models are commonly introduced to resolve the conflicts. These work to some extent but are rather costly and inefficient (Harrington 1995).

[1] The concept of conflict will be formally defined in Chapter 3.

Moreover, the traditional design approach usually requires designers to meet often in conference to resolve differences. This becomes impractical when the designers involved in the project are remote from one another. This situation is increasingly common in web-based design.

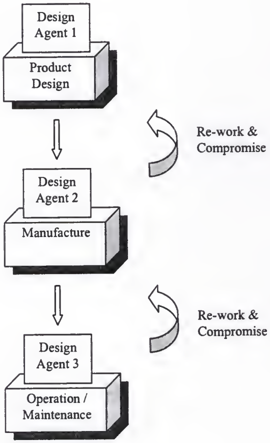


Figure 1-1: Traditional design

The concept of concurrent engineering (CE) has evolved to improve the design process. Concurrent engineering is a "systematic approach to creating a product design that simultaneously considers all elements of the product life cycle from conception through disposal" (Hoffman 1997). In a CE process, all the agents in a design team work together in the early design stage and important decisions are made considering the project's entire life cycle, rather than its individual design stages (Figure 1-2). This offers the potential for much more nearly optimal results than does the traditional sequential approach. However, to fulfill its promises, some serious problems in CE need to be resolved. One major problem involves explicitly dealing with a greatly increased number of disagreements and conflicts (Harrington 1996, Hoffman 1997). This problem becomes further exacerbated when design agents are located remote from each other, as is common in web-based design. Thus, it becomes important to ask how can potential conflicts be detected and avoided? When a conflict is encountered, how can it be resolved without seriously affecting the performance of the CE process? How much contribution can software agents make to dealing with conflicts? These questions are among the most critical issues actively studied currently in CE (Lander 1997a).

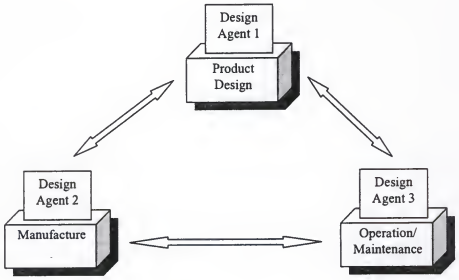


Figure 1-2: Concurrent Engineering Design

1.2 Conflict in Design

1.2.1 Existence of Conflict in Engineering Design

Most literature assumes that conflict exists commonly in the engineering design (Klein 1990, Lander 1997a). However, only a few empirical studies of real world conflict between designers in collaborative concurrent design have been found. For example, Fleischer did some empirical studies in a case of conflict between the design engineers and the manufacture engineers because of different design considerations (Fleischer 1997). In his study, the design engineer's primarily concern is product performance, while the manufacturing engineer cares more about ease of manufacturing with low cost. The different preference and priority of the two types of engineers lead to conflicts in the design process.

1.2.2 Definition of Conflict

Conflicts have been defined in a number of ways. For example, conflict can be defined as "disagreement between two or more viewpoints on some decision or values proposed in a design" (Pruitt 1981). Different opinions on design issues among agents commonly exist in the design process. The iterative exploration method (Easterbrook 1993), which serves as the heart of nearly all design, is a process for compromise and resolving disagreement. Often, it works reasonably well and the different designs converge into a compromise solution. However, at times resolution of disagreements is non-convergent, causing the design process to be blocked. Therefore, it becomes necessary to deal explicitly with the conflict.

The formal conflict definition used in this work will be discussed in Chapter 3.

1.2.3 Conflict in Traditional Design

Conflicts certainly exist in traditional design. However, in the traditional approach, agents often have poor communication and seldom discuss conflicts before major commitments have been made (Harrington 1996). Rather, the agents make their decisions based on their own local goals, resources, and situation. The conflicts are there, but implicit and hidden. When conflicts are finally recognized, it is often late in the design process, when it is difficult and expensive to resolve them. Agents have either to backtrack in the design process or to live with the flaw, neither of which is a good solution.

1.2.4 Conflict in Concurrent Design

In the CE environment, all the agents are involved in a project concurrently, and seek to communicate their experience, knowledge, priorities, and criteria to all other participants (Owen 1985). Conflict thus becomes much more explicit. All members involved in a design effort are expected to join in the discussion and negotiation over disagreements encountered. In the CE process, conflict management becomes a more complex and crucial task. There are many types of conflicts with various causes and operation mechanisms. Further, conflict management expertise commonly varies from one type of conflict to another (Landers 1997a). Thus, it is important to identify conflict domains and sources.

1.2.5 Conflict Domains and Sources

Generally, conflicts may be classified as a belonging in one of the two domains (Matta 1996),

1. Product Domain, the domain of the product being designed,
2. Process Control Domain, the domain of control and organization of design activities.

Conflicts in the process control domain are complicated, poorly understood, and rarely studied in previous research (Matta 1996). In this study, we focus on the product domain conflicts.

The primary sources of product domain conflicts are:

1. Misunderstandings caused by the designers' different beliefs, viewpoints, assumptions, terminology, tools and criteria for evaluating design objects (Ramesh 1994, Lander 1997a),
2. Conflicting opinions over the conditions and consequences under which the decisions of product design are made (Easterbrook 1993, Sycara 1991).

Meanwhile, process control conflicts may arise from the following causes,

1. Inconsistency in strategies and methods applied by design agents (Easterbrook 1993),
2. Different opinions on the allocation of tasks to each design agent (Easterbrook 1993),
3. Divergence of responsibilities and failure of cooperation among agents (Matta 1996).

The process control conflicts will not be considered further in this work.

1.2.6 Cooperation of Human and Software Agents

Humans play the principal role in most engineering design today and are likely to continue to do so far into the future. However, there are rarely enough resources to provide sufficient human Conflict Resolution (CR) expertise. Thus, the most valuable role for a software CR agent is believed to be supporting and augmenting the efforts of human design agents (Werkman 1991, Hu 1996). Therefore, in this work, focus is on developing a software agent, which can facilitate conflict avoidance and resolution by human agents. It is assumed that human agents are cooperative in the design process. That is, agents are assumed to be trying to cooperate.

CHAPTER 2
REVIEW OF RELATED WORK

Current research in CE design systems is commonly divided into three categories: information flow, concurrent design process, and conflict management. Here the first two are discussed only briefly and the third is emphasized.

2.1 Information Flow Management

Standards are developed to coordinate the data and communication in each stage of information flow. For example, Standard for the Exchange of Product (STEP) is proposed by Lin (Lin 1996) for enhancing computer interpretation of information on products. The CORBA approach (Mowbray 1995) and Knowledge Query & Manipulation Language (Finin 1994) are used to improve communication and interoperability among design agents.

2.2 Design Process Management

In CE processes, coordinating methods and expertise such as Computational Market Model (Wellman 1995) and Tracking Pareto Optimality (Petrie 1995) are applied to ensure that multiple agents coordinate their design activities to produce quality designs.

2.3 Conflict Management

Essential to performance of CE design processes, conflict management is clearly an important topic in current CE research. A number of approaches and techniques have been studied to detect, avoid and resolve CE conflicts efficiently.

2.3.1 Conflict Classification

It has been found that the methods used to detect and resolve one kind of conflict often will not work well for another one. Therefore, a common first step in conflict management is to build a taxonomy of conflict classes (Lander 1997a).

In Klein's cooperative design system (Klein 1991), a taxonomy of conflict classes associated with general advice suitable for resolving conflicts in these classes is proposed. Applying his conflict resolution (CR) expertise to local area network (LAN) design, Klein creates a taxonomy of four conflict types with a total of 115 conflict classes (Klein 1990, Klein 1991). Klein's taxonomy of conflict classes covers a broad range of CR problems. However, Klein does not clearly distinguish a Conflict from a Problem (or Difficult Issue) in the design. Conflict is the disagreement between two or more agents due to different design approach or parameter set-up etc. In contrast, Problem means an agent's difficulty in achieving the design goals. Also, Klein does not clearly distinguish between disagreements that are being successfully resolved by the basic design process and disagreements in which the basic design process is failing and thus is blocked. Furthermore, many of the classes are identified specifically to solve conflicts in LAN design and thus cannot be part of a generic CR system.

In contrast to Klein's flat structural organization of classes, which gives little information about the interaction between different classes, Matta builds a topology of conflict classes in a hierarchical structure (Matta 1996). In Matta's viewpoint, conflict is mainly handled as a disagreement between some participants rather than as a problem involving design and requirement. Two generic categories of conflicts are proposed by Matta: strategies and propositions. These two categories are then refined from very generic to very specific classes. In defining the hierarchy of conflict classes, the causes of various conflicts and their distinctive behavior are revealed and highlighted.

2.3.2 Conflict Detection and Avoidance

Probably the best way to manage a conflict is to avoid it before it happens (Lander 1997a). Since the majority of conflicts in CE systems are caused by misunderstanding and incomplete knowledge about the project design among agents, an information-sharing strategy is widely used in conflict avoidance. In this strategy, information such as local constraints, goals, and priorities are shared as much as possible among the agents involved in the design project (Lander 1997a).

In application, the detailed process of sharing information among agents affects greatly the performance of CR and the efficiency of design systems. Lander introduces the concept of meta-information, which describes some abstraction of an agent's solution space rather than its details (Lander 1997b). A reusable-agent system, a multi-agent computational system that uses a sharing meta-information mechanism and encodes some CR expertise, is applied to improve the efficiency of conflict avoidance. When required, the reusable agent can dynamically apply appropriate CR expertise to diverse applications to avoid conflicts.

Some approaches based on "personal" information sharing have also been studied. Ramesh suggests to share part of private information such as terminology used by each participant to avoid the conflict (Ramesh 1994). Rollinger works on improving communication among agents to minimize knowledge inconsistency and conflict (Rollinger 1996).

2.3.3 Conflict Resolution

Not all conflicts can be avoided in CE design processes. When a conflict occurs, appropriate conflict resolution (CR) expertise must be applied to resolve it efficiently.

Early Psychology and Social Science Approach

Early work on conflict resolution focuses on conflict in group interaction and human participants' behavior. Knowledge and expertise from artificial intelligence and social sciences, especially psychology, are heavily utilized. For example, Fedrizzi proposes an interactive multi-user group decision support system (GDSS) using fuzzy logic and linguistic knowledge (Fedrizzi 1988). A limitation of this approach is that it focuses on some very specific social or psychological issues rather than generic CR strategies and methods.

Computational Model Approach

Computational models, which encode the CR expertise by using mathematical formalism, can be introduced to create a software agent to detect and resolve the conflicts in CE design process.

One of the early approaches to computational CR methods is the Development-Time CR approach. Like compiling programs in software development, this approach attempts to resolve all potential conflicts by thorough discussion in the early stage before actually running the design process. A rule-based expert system is often used to coordinate the CR in multi-agent design. The compiling process includes verifying completeness and consistency of the expert system (Suwa 1982, Benzem 1987).

The Development-Time CR approach is static and impractical since it is virtually impossible to anticipate all the conflicts early in a design process. When new conflicts are encountered, the re-compiling process becomes fairly time-consuming. To overcome its shortcomings, some Running-Time CR approaches are later introduced to allow the conflict to be assessed when the system runs and resolved by tracking back the design process or relaxing some constraints (Feldman 1985). These approaches incorporate limited CR expertise. The CR expertise is represented by restrictive mathematical formalisms, which is one of the main disadvantages of such approaches.

To improve the Running-Time approach, Klein introduces a taxonomy of conflict classes and does some substantial research work on generic CR expertise (Klein 1990, Klein 1991). In his Cooperative Design System, Klein builds a taxonomy of conflict classes. Associated expertise is developed to resolve both generic conflicts and conflict in a specific domain such as Solar Home design and Local Area Network (LAN) design. In Klein's system, a hierarchy of conflict classes is built, in which each conflict class inherits the attributes from its parent class and has more domain-specific features associated with more specific CR expertise. When a conflict is encountered, the design agent, which is a software agent, will determine the type of conflict and the appropriate

CR method from the very generic to the very specific to finally resolve the conflict. While the model works well in LAN design, many of the CR methods proposed are too specific to the LAN design and have limited use as generic CR guidelines.

Recently, Sobieski and Haftka introduce response surfaces, a low-order polynomial, and neural networks to represent disciplinary response and objectives to an overall system designer, and facilitate conflict resolution (Sobieski 1996). These response surface and neural network methods have been applied to conflict resolution in design studies of supersonic transports (Balabonov 1996, Giunta 1996).

Negotiation Approach

Rather than depending on the software agent to completely handle the conflict resolution, some suggest using negotiation between human agents with aid from some kind of AI system to resolve the conflicts and reach consensus in the design process. Matta suggests defining a library of generic components to guide CR modeling in CE design systems (Matta 1996). These components are arranged in a hierarchical structure, from the very generic types to the domain-specific ones. Meanwhile, various negotiation methods such as locating a consensus and introducing a third party are studied and encoded into the library. The negotiation approach is also applied by Findler in coordinating multiple agents in a Distributed Artificial Intelligence (DAI) system, which is a collection of geographically distributed intelligent decision-making agents with limited shared resources (Findler 1995). DAI based negotiation is also studied by Werkman in the Designer Fabricator Interpreter (DFI) system that is developed to foster interaction and improve communication among agents from diverse background such as preliminary designers, engineers and fabricators in order to solve conflicts efficiently (Werkman 1991).

2.4 Summary

In previous study, much work has been done in conflict classification, avoidance and resolution. However, there are some topics that have not been explored in depth yet. For example, how to separate the conflict from the difficult problem in design? How to identify the conflict causes in a web-based CE design environment? These questions will be studied in this work. In the next chapter, we will first explore the nature of conflicts.

CHAPTER 3
THE NATURE OF CONFLICTS

3.1 Conflict Definition

Pruitt (Pruitt 1981) and Harrington (Harrington 1995) define a conflict as disagreement between two or more viewpoints on some decision or values proposed in a design. These authors distinguish conflict from the internal design problem inside each design agent. However, disagreement does not always lead to conflict. In routine design processes, the iterative exploration method, i.e. negotiation, is commonly used to find alternatives that are accepted by all parties (Easterbrook 1993). In the present work, conflict is considered as an exception to the normal routine design, where the disagreement resolution is non-convergent, causing the design process to be blocked.

Also, requirements play an important role in the existence of conflict. Without some related requirement, no conflict can exist; even thought agents disagree on a design issue. For example, two automobile design agents, one in charge of exterior and another in charge of interior have different opinions on the color selection. The exterior design agent prefers black, while the interior agent likes red. A conflict emerges only if there is a requirement of compatibility of interior and exterior colors. Otherwise, the conflict simply does not exist.

Based on the above insights, a conflict is defined here as follows: A conflict is a disagreement, between two or more agents involved in achieving some requirement, which is not being resolved satisfactorily by the basic design process.

3.2 Conflict Detection

It has been suggested that the best way to handle a conflict is to detect and avoid it (Lander 1997a). Various methods have been studied in previous research. For example, Lander proposed a sharing meta-information strategy (Lander 1997a).

R. Zhuang developed a computational model, which uses public information such as the design parameter values to detect the conflicts and problems in design (Zhuang 1999).

Conflict detection is not the focus of this study. Rather, Zhuang's model will be used to detect conflicts in design processes and provide input to the software Conflict Cause Identification (C^2I) agent developed in this study. Since Zhuang's model cannot distinguish conflicts from problems, a computational model based on Value Indicator Patterns (VIP) and requirement types is developed and used in the C^2I agent to separate conflicts from problems in design processes. The details of the computational model and the C^2I agent will be discussed in Chapter 4 and 5.

3.3 Conflict Classification

Previous studies show that conflicts vary from each other in fundamental causes and behaviors (Klein 1991). Conflict resolution methods tend to be appropriate for specific kinds of conflicts. Often, one method used to resolve one kind of conflict does not work well for another one. Thus, it is desirable to group the conflicts in different categories, which are called Classes in this study.

3.3.1 Dimensions of Conflict Classes

As mentioned in Chapter 1, a conflict may generally be considered as belonging to one of the two domains, i.e. design product domain and process control domain (Matta 1996). Because the process control domain conflicts are complicated and little understood (Matta 1996), only the design product domain conflicts are studied in this work.

A design product conflict (simply called conflict hereafter) may be described along two dimensions: requirement type and fundamental cause (Figure 3-1).

1. Requirement type: Requirement and conflict are inseparable in engineering design. The conflict will not exist if there is no related requirement. Also, interactive relations and requirement type play an important role in the conflict characteristic and behavior (Lander 1997a). Studying the various requirement types should help the design agent to understand the conflict behavior and apply appropriate conflict resolution methods to a specific conflict. In this study, we classify the requirements into three types[1] as following:

   1.1 Aggregated requirement,

   1.2 Interface related requirement,

   1.3 Functional influenced requirement.

[1] Detailed definition of these requirement types will be given in Section 3.4.

2. Fundamental cause: Previous studies indicate that conflicts may occur for various reasons, such as misunderstanding in communication, local-goal incompatibility (Rollinger 1996), and different criteria for evaluating alternative designs (Lander 1997b). Finding the conflict causes is the key to conflict resolution methods. Here, we classify the conflict fundamental causes into following categories[2]:

   2.1 Different terminology,

   2.2 Different perception of the design object,

   2.3 Conflicting local goals,

   2.4 Inadequate communication.

Thus, the conflict class is a combination of requirement type and conflict fundamental requirement. As Figure 3-1 shows, a conflict class may be represented as a cell in the two-dimensional matrix.

For example, conflict class X may be expressed as:

$$\left(D_x^1, D_x^2\right) \tag{3-1}$$

Where $D_x{}^1$ is the requirement type dimension,

$D_x{}^2$ is the fundamental cause dimension.

In conflict class identification discussed in following Chapter 4, it is found that different requirement types and fundamental causes lead to various different design behaviors, i.e. value indicator patterns. The two-dimensional matrix representation clearly illustrates the components of conflict classes.

---

2 Detailed definition of these fundamental causes will be given in Section 3.4.

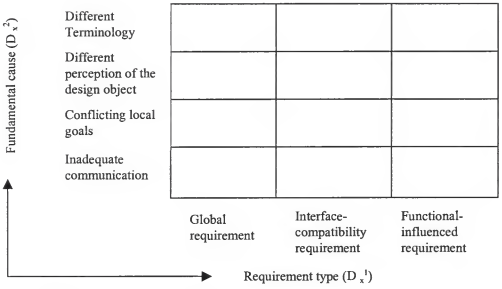

Fundamental cause (D_x^2)
Different Terminology
Different perception of the design object
Conflicting local goals
Inadequate communication
Global requirement
Interface-compatibility requirement
Functional-influenced requirement
Requirement type (D_x^1)

Figure 3-1: Dimensions in conflict class

3.3.2 Additional Characteristics of Conflicts

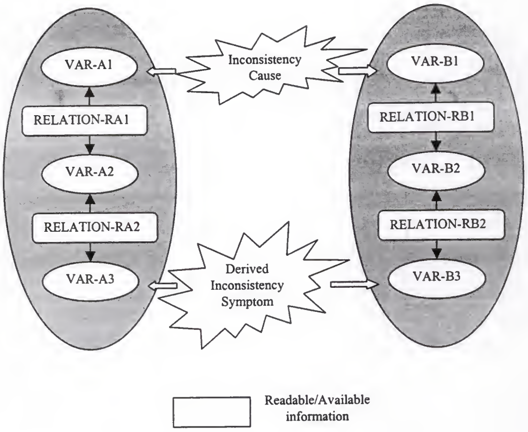
In this section, two important additional characteristics of conflicts are discussed: (a) the availability of variable values and relationships to design agents involved in the conflict, and (b) the balance of variables involved in the conflict.

Explicit Logical Inconsistency vs. Implicit Logical Inconsistency

In engineering design, conflicts are the basic inconsistency causes, which result in the derived inconsistent symptoms throughout the relations among modules, components and their attributes. For example, consider two modules designed by two agents, i.e. Agent-A and Agent-B in a design project (Figure 3-2a and Figure 3-2b). Each module contains a set of attributes, connected by intra-module and inter-module relations. Some of the variables and relations are public, that is, besides their own agents; they are accessible to other design agents and the C^2I software agent. Others are internal, and accessible only to their own agents. In design process, the values of variables VAR-A1 and VAR-B1 are considered inconsistent if they lead to explicit inconsistencies. Thus, an inconsistency between VAR-A1 and VAR-B1 propagates along relation chains in both modules, leading to an observable explicit inconsistent symptom, which involves two entirely different variables VAR-A3 and VAR-B3.

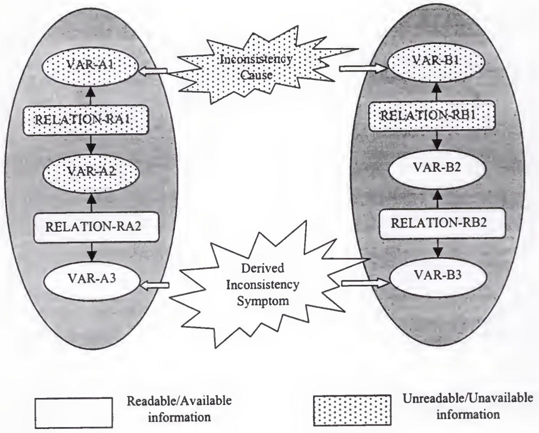
To determine the conflict class, the software agent needs to backtrack from the derived explicit inconsistency symptom to the basic inconsistency cause, through the linking relations. In some cases, these relations are available and readable to the human and software agents. But in other cases, they are inaccessible because they are part of the design agent's private information or hidden within the designer's mind, perhaps even unexpressed verbally. In the first set, named Explicit Logical Inconsistency, all associated relations and variable values are accessible and readable to all agents (Figure 3-2). In the second set, part or all of the relations and variables are inaccessible or unreadable. These are called Implicit Logical Inconsistency (Figure 3-2).

While the implicit logical inconsistency exists commonly in real-world engineering design, it is rarely studied because of difficulty in tracking the privately held information in design and will not be studied here. In this study, only conflicts of the explicit logical inconsistency type are studied.



(a) Explicit logical inconsistency

Figure 3-2: An example involving (a) explicit logical inconsistency (b) implicit logical inconsistency



(b) Implicit logical inconsistency

Figure 3-2--continued

Balance of Attributes

Attribute balance may vary in different requirement types. In global requirements, attributes determined by all involved parties are balanced in that they are independent and directly contribute to the value of the global requirement variable. In the interface-related requirements, the related attributes are required to be compatible, i.e. of the same value in some sense. Thus, the attributes are balanced. However, in the functional-influenced requirements, the relation between attributes is one-directional, i.e. one attribute's value is determined from the value of another one through some functional relationships. In this case, the attributes are referred as unbalanced.

Study of attribute balance helps us to understand the interactive relations and behavior of attributes in various requirements. It is useful in identification of requirement types in conflict class identification.

3.4 Conflict Class Structure

From the previous sections, it is concluded that the most basic characteristics of the conflicts to be studied are requirement type and fundamental cause. Thus, conflict class is defined as a combination of requirement type and fundamental cause, as Figure 3-3 shows.

The requirements are grouped into set level and individual/local level. At set level, there is aggregated requirement, while at individual/local level; there are interface compatibility requirements and functional-influenced requirements.

Similarly, the fundamental causes are grouped into three categories, i.e. different perceptions, conflicting local goals, and inadequate communication. The conflict causes in different perceptions category are further divided into five cases, i.e. different terminology, different point of view, different information, different model/tool, and different relational definition of an attribute.

In the following section, the definitions and examples of these conflict classes are discussed in detail.

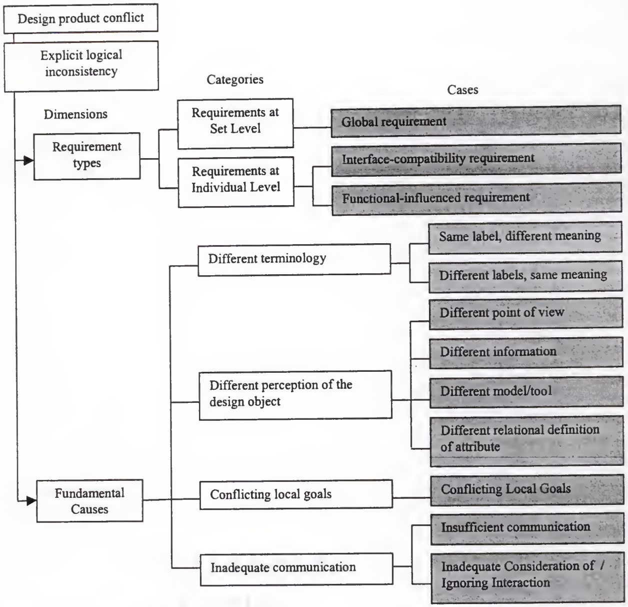

Design product conflict
Explicit logical inconsistency
Dimensions
Categories
Cases
Requirement types
Requirements at Set Level
Requirements at Individual Level
Global requirement
Interface-compatibility requirement
Functional-influenced requirement
Different terminology
Same label, different meaning
Different labels, same meaning
Different perception of the design object
Different point of view
Different information
Different model/tool
Different relational definition of attribute
Fundamental Causes
Conflicting local goals
Conflicting Local Goals
Inadequate communication
Insufficient communication
Inadequate Consideration of / Ignoring Interaction

Figure 3-3: Conflict class structure

3.4.1 Requirement Types

Global Requirement

Definition. A global requirement is a requirement that is imposed on a global variable, whose value is determined by two or more independent variables from different modules or parts. All the involved module variables are balanced.

Examples. Global requirements can be divided into three categories: simple additive aggregation, complex function aggregation, and maximum/minimum requirement, as shown in the following examples.

1. Simple additive aggregation: In some aggregated requirement, the value of the variable on which the requirement is imposed is simply aggregated by the sum of the contributing variables. For example, the overall weight of a plane is the sum of the weight of its modules:

$$Plane.Weight = \sum_{AllModules} Module.Weight$$

2. Complex function aggregation: Some requirement-related variables, such as aircraft speed and range, are determined by complex functions of associated module variables. These functions are often non-additive and non-linear. For example, the speed and range of a plane are complex functions of parameters of its configuration, propulsion, and fuel tank modules:

$$Plane.Speed = ComplexFunction_1\begin{pmatrix} Propulsion.Thrust, \\ Configuration.Drag,... \end{pmatrix}$$

$$Plane.Range = ComplexFunction_2\begin{pmatrix} Plane.Speed, \\ Propulsion.FuelRate, \\ FuelTank.Fuel,... \end{pmatrix}$$

3. Minimum/Maximum requirement: In this case, the requirement-imposed variable is actually determined by the minimum or maximum value of the associated module/component variables. For example, the strength of a plane is the minimum strength of all modules and attachment components:

$$Plane.Strength = Min(Module.Strength, AttachmentComponent.Strength)$$

Interface-Compatibility Requirement

Definition. The interface-compatibility requirement is used to enforce the compatibility between module/component variables involved in an interface between modules. Since the involved modular variables are required to be compatible or of the same value, they are balanced in the requirement relation.

Examples.

1. Interface-compatibility requirements are commonly used in industry such as network engineering. In Klein's LAN design (Klein 1990), the data transferred between two stations is required to have the same coding format.
2. In aircraft design, interfaces exist between attached modules, i.e. Propulsion and Wing Structure. We assume these modules are fastened by bolts. Thus, the holes for installing bolts and nuts of two adjacent modules are required to be in the same position.

Functional-Influenced Requirement

Definition. A requirement of this type is imposed on a module's variable, which is determined by another module's variable value, or several other modules' variable values, shown in Figure 3-4. Clearly, the variables in the functional-influenced requirement are unbalanced.

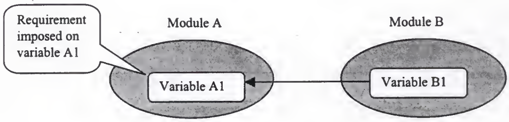


Figure 3-4: An example of functional-influenced requirement

Examples.

1. In Klein's house design case (Klein 1991), there is a requirement on the cooling cost which is determined by a number of external features:

   $House.CoolingCost < \$50 / month$

   The cooling cost is determined by the energy required by the cooling system, which is a complex function of outdoor temperature, sunlight, and the layout of the windows especially in the south side[3]:

   $House.CoolingCost = Function_{K1}(House.CoolingPowerOutput)$

   $$House.CoolingPowerOutput = Function_{K2}\begin{pmatrix} Room.Temperature, \\ Environment.Temperature, \\ Environment.Sunlight, \\ SouthSide.WindowLayout,... \end{pmatrix}$$

[3] In the functions, subscripts K1 and K2 indicate these are functions referred from Klein' work.

The sunlight effects the cooling cost greatly. In summer, the high summer sun raises the outdoor temperature. Also, if the south-facing front facets have large windows designed for better appearance, the insolation through these windows will increase the room temperature and lead to excessive cooling cost. Thus, the cooling cost of a house is functional-influenced by the appearance, i.e. the layout of its windows of the south-facing facet. The requirement imposed on the cooling cost is of functional-influenced type.

2. In aircraft design, the loading on a structure module is required to be less than some fraction of its predicted strength:

   $Structure.Loading < Structure.Strength$

   For the rear structure, i.e. horizontal tail, its loading is a complex function of environmental factors such as air pressure, temperature, vibration, and noise.

   $$Structure.Loading = Function_1\begin{pmatrix} AirPressure, \\ Temperature, \\ Vibration, Noise, ... \end{pmatrix}$$

   Meanwhile, the environmental vibration and noise are affected by the propulsion module's location, exhaust, and noise level.

   $$Vibration = Function_2\begin{pmatrix} Propulsion.Location, \\ Propulsion.Noise, ... \end{pmatrix}$$

   Thus, the loading on a rear structure module is influenced by the propulsion module's attributes (i.e. location, noise) via a chain of related functions. The requirement over the structure strength and loading is of functional-influenced type.

3.4.2 Fundamental Causes

Fundamental causes of conflicts are grouped into four categories: different terminology, different perceptions of the design object, conflicting local goal, and inadequate communication.

Different Terminology

Definition. In engineering design, agents use labels, which are also called terms, to represent the meaning of design entities such as variables and relations. In the case of different terminology, agents have different label-meaning relations for some variables in design.

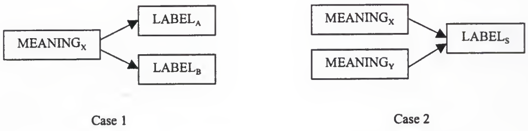


Figure 3-5: Two cases of different terminology

This may involve two cases (Figure 3-5):

1. Agent-A and Agent-B use two different labels, i.e. LABEL_A and LABEL_B to represent the same meaning, i.e. MEANING_X. Thus, the two agents get confused in communication, not knowing they are actually talking about the same thing since labels they use are different.
2. Agent-A and Agent-B use a same label, i.e. LABEL_s to represent two different concepts (i.e. meanings), i.e. MEANING_X and MEANING_Y. In this case, each agent assumes that the other one talks about the same thing as he does.

Inevitably, each agent will be misled by the unexpected conclusion of the other.

Examples. Both above situations occur in the engineering design. The following examples illustrate the different terminology concept.

1. Multidisciplinary product design: Designing multidisciplinary products such as DVD (digital versatile disc/digital video disc) players often involves various specialists in the fields of mechanical, electrical, optical, and software engineering. Sharing information becomes complicated by the different terminology, conventions used by each specialist (Ozawa 1998).
2. Structure strength: In structure design, the designers may have different understanding of the structure strength, i.e. static vs. dynamic. From a static perception, compression buckling and extensional strength are likely to be the main concerns. However, from a dynamic perception, strength may mean the ability to resist fatigue and fracture caused by vibration, impact and noise. Design agents with these two different backgrounds may have trouble in understanding each other because of different meanings of terminology.
3. Cargo capacity of an airplane may have the following different meanings:
    - Cargo weight, i.e. how many pounds of cargo the plane can carry,
    - Volume of cargo cabin, i.e. how many cubic feet of cargo can be put into the cabin,
    - Minimum width and height of the cargo cabin, i.e. how large a single cargo can be put into the plane.

Thus, people with different interests and backgrounds may interpret the term "cargo capacity" in different ways, leading to trouble in information sharing and design process.

Different Perceptions of the Design Object

In collaborative design, the agents involved often come from various engineering backgrounds with different knowledge, experience, priorities, and criteria. Thus, it is common they may have different perceptions of the design entities, i.e. they may have different point of view (Ramesh 1994, Lander 1997a). Here, the different perceptions are divided into following cases: (1) different point of view, (2) different information, (3) different model/tool, and (4) different functional definition of an attribute. Definitions and examples of each case follow.

Different Point of View

Definition. In collaborative design, agents with different knowledge and experience background may consider the same object from different viewpoints, leading to different understanding and conclusion.

Examples. Conflicts resulting from different points of view are common in engineering design (Ramesh 1994, Lander 1997a). The following are some examples:

1. Automobile design: in the automobile industry, there are different points of view about good car performance, that is, how to balance various primary and auxiliary functions. Some companies prefer adding more auxiliary features such as leather interior, and climate control. Obviously, it brings more fun to driving. However, to keep the cost low, some primary features such as

reliability and fuel efficiency may be sacrificed. Other companies tend to focus on the primary functions such as the reliability and fuel efficiency of the engine and drive train system. In these designs, the auxiliary features are considered as luxury or optional. These cars run well, lasting longer, yet are not as comfortable as the former designs. When design agents with these two points of view work together, they can be expected to have conflicting opinions in design process.

2. Military design vs. civilian design: In aerospace industry, the military aircraft designer's point of view is often dramatically different from that of the civilian aircraft designer. Traditionally, in military aircraft design, high performance (i.e. agility, speed) is often the highest priority, while cost and maintenance are secondary concerns. In contrast, civilian aircraft are designed to be less expensive in both manufacturing and maintenance. Hence, performance factors, such as agility and speed, are not the dominant concerns. The different philosophies may lead to different point of view to the same design object. For example, in plane structure design, the military-background agents incline a design that is super light and super strong, being able to sustain loading in pulling high G. The cost may be very high. The civilian-background agents prefer a conventional design, which is easy to build, and cost saving. It does not have to be super strong, i.e. only being able to sustain relatively lower loading than that of the military aircraft. When the agents with different background work together, different point of view may lead to conflicts.

3. Aerodynamic vs. structure in aircraft design. Two agents work closely together on design of the plane's wing. One is in charge of the aerodynamic configuration and another works on the structure part. In the aerodynamic point of view, afterward-swept wing with high aspect ratio is desirable. It provides high lift and low drag for the subsonic transport plane. However, in the structure point of view, straight wing with low aspect ratio is stronger and lighter. Based on different consideration, these two agents may have conflicting opinions on the wing design (Figure 3-6).

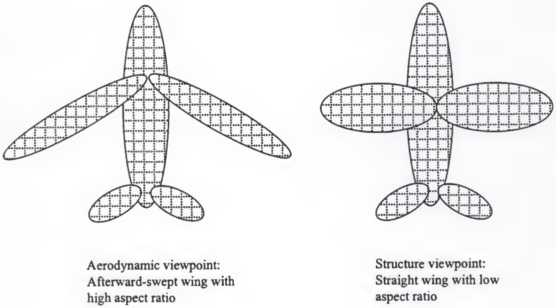


Figure 3-6: Different point of view on the wing configuration in aircraft design

Different Information

Definition. In collaborative design, two design agents sometimes use different information, input such as initial value, make the same calculation, but get different results.

Examples.

1. Numeric analysis: Different information input often causes conflicting results in the numeric analysis in design. For example, two design agents use the bisection method to compute the roots of a polynomial equation. If they input different initial ranges, the roots found may be different even though the computing processes are same. Here, as Figure 3-5 shows, Agent-A starts with range [A_1, A_2], while Agent-B begins with range [B_1, B_2]. Also, we have $[A_1, A_2] \cap [B_1, B_2] = \phi$. Using bisection method, Agent-A may get root R_A and Agent-B may get root R_B where $R_A = R_1$ and $R_B = R_3$ (Figure 3-7).

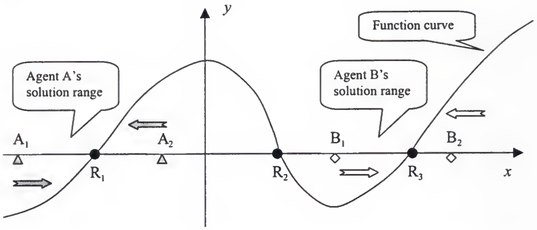


Figure 3-7: An example that different initial ranges lead to different solutions

Different Model/Tool

Definition. Based on various considerations, i.e. experience, understanding, and availability of tools, two design agents use different models or tools to describe or evaluate the design object. Often, it leads to different result.

Examples.

1. Different models – Elastic model vs. viscous elastic model: In industry, elastic model and viscous elastic model are often used to analyze the strength of composite structures. Some people prefer the elastic model, which is simple and easy to use, while others like the more complicated viscous elastic model. In some cases, i.e. sandwich structure whose viscous characteristics becomes non-negligible, the computed results from the two models would be different in some ways. Agents using different models could have conflicts in their results.
2. Different tools – NASTRAN and ABAQUS: NASTRAN and ABAQUS are two common FEM (finite element method) tools for structure analysis. Some companies license NASTRAN, while others purchase ABAQUS. Generally, their functions are similar with each other. However, it is learned that the two software programs incorporate different models and algorithms in some cases[4]. Thus, the computed results from the two software programs may become different.

[4] From a private communication with Dr. Bhavani V. Sankar, Professor of Department of Aerospace Engineering, Mechanics, and Engineering Science, University of Florida.

For example, two groups participate in a joint project of structure design. Agents in one group use NASTRAN, while agents in another group use ABAQUS. Agents in the two groups may have conflict in their results of structure analysis.

Different Functional Definition of an Attribute

Definition. In this case, two agents use different functional descriptions of attributes with the same labels.

Examples.

1. Overall project cost: The overall project cost (OPC) is an important attribute concerned in evaluating a project. However, there are various possible functional definitions of this attribute. Traditionally, OPC contains only the development and manufacturing cost.

   $$Cost_{Overall\,Project} = Cost_{Development} + Cost_{manufacturing}$$

   Since the 1980s, more and more engineers tend to include the maintenance cost, considering OPC as project life-cycle cost:

   $$Cost_{Overall\,Project} = Cost_{Development} + Cost_{manufacturing} + Cost_{maintenance}$$

   Meanwhile, for some specific products such as batteries, the disposal cost is also included in the OPC:

   $$Cost_{Overall\,Project} = Cost_{Development} + Cost_{manufacturing} + Cost_{maintenance} + Cost_{disposal}$$

   If the agents involved in the same project have different functional definition of attributes as above example, they may have conflicts in the design process.

Conflicting Local Goals

Definition. In a design team, each agent has his local goals. Some local goals are explicit (i.e. the thrust goal for a propulsion agent), while others are implicit (i.e. personal goals). This study focuses only on the explicit local goals.

To achieve their local goals, design agents manipulate the modules' variables in design. In some cases, their local goals are conflicting with each other, leading to violation of the associated requirements.

Examples.

1. Positive feedback loop: One example of conflicting local goal is the positive feedback loop. In this case, an agent increases his attribute value in pursuit of his own local goal. This in turn causes another agent to increase his attribute value. In consequence, the related agents have to keep on increasing their attribute values in cascade. As a result, the global requirement is violated. For example, in tank design, the overall performance of a tank is determined by three major factors, i.e. firepower, mobility, and protection (Hunnicutt 1988). The tank design often involves modules of gunnery (including fire-control system), power package (simply called engine hereafter), armor, and vehicle structure (simply called structure hereafter). As Table 3-1 shows, each module's agent has his own local goals. These local goals are often related to the module weight by functions, shown in Table 3-2.

Table 3-1: Local goals and related module attributes

AGENTS	LOCAL GOAL VARIABLE	LOCAL GOAL
Gunnery	Caliber (Cal) and length-caliber ratio (Len) of the gun	Bigger gun with larger caliber and length-caliber ratio for superior firepower
Structure	Internal space volume (Vol) and structure strength (Strength)	Large internal space (for guns, engines, fuel) and more structure strength
Armor	Armor protection area (Area)	More armor to cover more area of the tank
Engine	Horse power (Hp) and fuel (Fuel)	More powerful engine for better mobility, more fuel for range

Table 3-2: Local goals and related module weight

MODULE AGENTS	LOCAL GOAL VARIABLE	RELATED TO MODULE WEIGHT
Gunnery	Caliber (Cal) and length-caliber ratio (Len) of the gun	$Weight_{Gun} = Func_A\begin{pmatrix} Cal_{Gun}, \\ Len_{Gun} \end{pmatrix}$
Structure (Struct)	Internal space volume (Vol) and structure strength (Strength)	$Weight_{Struct} = Func_B\begin{pmatrix} Vol_{Struct}, \\ Strength_{Struct}, \\ Weight_{Gun}, \\ Weight_{Engine}, \\ Fuel_{Engine} \end{pmatrix}$
Armor	Armor protection area (Area)	$Weight_{Armor} = Func_C\begin{pmatrix} Area_{Armor}, \\ Vol_{Struct} \end{pmatrix}$
Engine	Horse power (Hp) and fuel (Fuel)	$Weight_{Engine} = Func_D\begin{pmatrix} Hp_{Engine}, \\ Fuel_{Engine}, \\ Weight_{Structure}, \\ Weight_{Armor} \end{pmatrix}$

In pursuing these local goals, the agents may fall into a positive feedback loop, leading to violation of some global requirement such as overall weight of the vehicle. As Figure 3-8 shows, in the design, the gunnery agent attempts to field a bigger gun for superior firepower. To adapt the bigger gun, the structure agent tends to build a larger and stronger vehicle. Meanwhile, in order to provide better protection to the larger vehicle, the armor agent has to cover more area. Also, a more powerful engine, which is bigger, heavier, and less fuel-efficient, is needed. To contain the bigger engine and more fuel, the structure agent has to increase the vehicle size again, which often triggers another loop of design. All these come with the penalty of increased weight. As a result, the overall weight of the tank increases steadily, violating the global requirement of the tank weight. In history, the over-weighted tank design such as German Tiger-II in WWII, which resulted in immobility, often led to poor performance in battlefields (Macksey 1988). Another example is the American heavy tank program in WWII. After years of developed, the M6 tank was canceled by the Army due to its immense size and weight (i.e. over 60 tons, twice as heavy as the ones in service at that time) (Hunnicutt 1988).

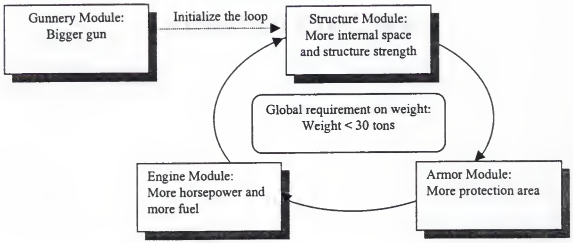


Figure 3-8: An example of positive feedback loop in tank design

Inadequate Communication

Definition. In this case, for a period of time in design, some agents do not have sufficient communication necessary for coordinating their designs, or keep on ignoring the requests from other agents and the changes on the related attributes by other agents.

Example.

1. Airplane design commonly involves multiple teams responsible for different parts (i.e. aerodynamic configuration, structure, propulsion, fuel tank, cargo etc.). These different parts are the responsibility of different human design agents. Human agents tend to work on the design issues one by one. That is, each agent will focus on one aspect at a time, and give significant effort to do it before going on to the next issue. As long as agents pay prompt attention on their colleague's work, this will not cause any conflict. However, some agents may totally focus on their own design, ignoring the related variable changes or

the requests from other agents. Such situation may lead to conflicts in design process. For instance, the delay in communication may lead to agents using obsolete information in design.

3.5 Conflict Cause Identification

With knowledge of conflict nature discussed above, the next task is to develop an appropriate method to identify the conflict class and cause when a conflict is detected in the design process. The details of Conflict Cause Identification are discussed in Chapter 4 and 5.

CHAPTER 4
CONFLICT CAUSE IDENTIFICATION

Effective identification of conflict cause is important for resolving conflicts in design processes. It facilitates the design agents to assess the situation, i.e. the cause and behavior of the conflict precisely, and thus helps the agents to apply appropriate methods to resolve the conflicts.

4.1 Introduction: Identification of the Conflict Causes

4.1.1 Review of Previous Work

Historically, various approaches have been proposed to identify conflict causes.

In Klein's work (Klein 1990, Klein 1991), a set of preconditions is suggested for each conflict class[1]. In the design process, the conflict resolution (CR) agent queries the design agents using relatively abstract questions in order to find a match to a conflict class's precondition. A specific query language is developed to carry out the query process. While formal language-based query is conceptually valid and makes the identification process well structured, its application is limited in real world design. In general, design agents prefer communicating in natural language. The formal query language is often hard to understand and therefore avoided by design agents (Easterbrook

[1] In Klein's work, conflict class is considered equivalent as the conflict cause.

1993). In addition, many of the conflict classes proposed by Klein are too specific to LAN design and have limited use for conflict resolution in generic situations. Furthermore, Klein's system does not distinguish conflicts from difficult problems in design.

Rather than using a query approach, Cointe develops a library of generic components, which stores the conflict-related design information (Cointe 1997)[2]. In this library, the design information is grouped into conflict class-related categories, i.e. needs, resources, and requirements. In finding the conflict class, the Evaluation agents assess the variables and relations in each category, looking for inconsistencies. Cointe's software agent works autonomously; that is, it retrieves the design information from the public data storage system such as the library rather than depending on querying the human agents, which they believe is often unreliable. However, in Cointe's system, the clustering of information in the library is based on various design facets rather than different conflict behaviors. In addition, the Evaluation agent only utilizes the current value of design data, and ignores the history of these data.

4.1.2 The Basic Approach Taken in This Work

Based on previous studies and our own research, a web-based Conflict Cause Identification (C^2I) system has been developed. The C^2I system tracks the history of design information (i.e. variable values in a design entity), extracts the Value Indicator Patterns (VIPs) from the information, retrieve the requirement types, applies appropriate computation to determine the conflict causes, and communicates the results to the appropriate design agents, thus, supporting them in conflict resolution in the distributed design system.

The web-based C^2I system involves three major components: the conflict-related Agent Behavior Model, the Value Indicator Pattern, and the Design Process Graph.

Conflict-related Agent Behavior Model (ABM)

The conflict-related Agent Behavior Model (simply called the Agent Behavior Model or ABM hereafter) is a set of rules, which describe how human design agents behave and interact with each other in a web-based design environment. The ABM serves as the basis of Value Indicator Pattern and Design Process Graph analysis.

A general ABM would involve rules of agent behavior in various designing aspects. In this study, focus is on the conflict-related rules only. The agent behavior rules (ABR) are collected and created from both studies referenced and our experience in the Plane World simulation. Since these rules have not been rigorously validated empirically, they are considered as assumptions in this work.

Value Indicator Pattern (VIP)

Generally speaking, the pattern of an entity is a set of characterizing parameters, which describe some of its specific features (Wolff 1983). Pattern recognition and analysis is a popular data analysis method that identifies the patterns in the experimental data produced in an investigation and draws useful conclusions from them. In this study, Value Indicator Pattern (VIP) is viewed as a set of parameters characterizing variable

[2] Cointe's work also considers conflict class as equivalent to conflict cause, with no consideration of the requirement type.

values, collected in the design history, which are tracked and analyzed to determine the conflict causes in design process.

Here focus is on the public variable values and relations, which are published into the public database system in the design process and readable to all design agents and the software C^2I agent. In engineering design, there is also some information privately kept by individual design agent for internal use. Since it cannot be accessed by other agents, such private information is not included in the C^2I process.

Design Process Graph (DPG)

Using heuristic expert rules, expert systems are widely applied in industry and medical service to deal with difficult problems in complex domains which often resist precise description and rigorous analysis (Hayes-Roth 1983). One of the classical examples is MYCIN (Hayes-Roth 1983, Buchanan 1984), a rule-oriented expert system that addresses the problem of diagnosing and treating infectious blood diseases, which has been well established and tested for years.

Figure 4-1 shows the architecture of MYCIN. In diagnosing infectious blood diseases, MYCIN applies a set of heuristic rules to describe the relations of disease causes and consequent symptoms in various stages and situations. Using these rules, decision trees are constructed to describe the complex procedures from a fundamental disease to its corresponding symptoms, the cause-consequence relations involved and their possibilities, shown by the gray arrows. In reverse, the inverse processes are applied to backtrack the fundamental disease from the symptoms observed, shown by the dotted arrows (Buchanan 1984).

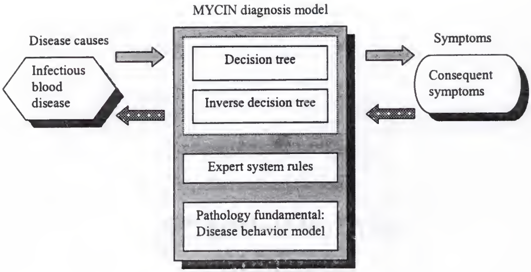


Figure 4-1: Diagnosis of blood diseases in MYCIN expert system

Using the MYCIN approach as a guide, a rule-based C^2I system has been developed, which uses the Agent Behavior Rules (ABRs) and resulting Design Process Graphs (DPGs) as the basis for identifying conflict class and cause from the Value Indicator Patterns (VIPs) obtained in design processes.

In the C^2I system, the DPG derived from the ABRs can be used to describe the complex mapping relations between the conflict causes, requirement types and the corresponding symptoms, i.e. Value Indicator Patterns in the design process. In the reverse process, the software C^2I agent can utilize the mapping relations described by DPG to identify the conflict causes from the requirement type available and VIPs observed, thus, supporting the Conflict Cause Identification. Because there is no general methods to invert the ABM rules in present artificial intelligence research[3], the mapping relationships between conflict classes and associated VIPs are used in the inverse identification process.

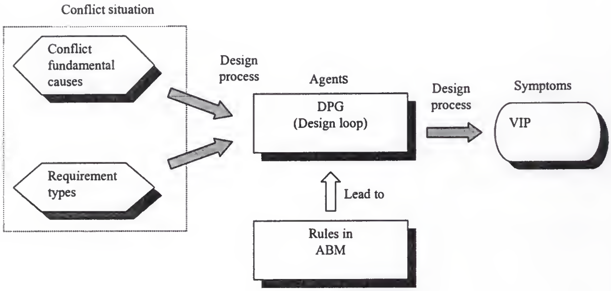
4.1.3 Conflict Cause Identification Model

The Conflict Cause Identification (C^2I) model, shown in Figure 4-2, involves five major parts, i.e. conflict situation, Agent Behavior Model, Design Process Graph, Inverse Identification Process, and symptom.

1. Conflict situations: Grouped by various classes, the conflict situation is a combination of fundamental causes and requirement types. In engineering design, the conflict situations are inputs to the design system, influence the agents, and produce the conflicting design results, i.e. the symptoms. These are originally unknown and the result sought by the C^2I system.
2. Agent Behavior Model (ABM): The model of conflict-related agent behavior describes how design agents behave in conflict-related situations. ABM forms the basis for the Design Process Graph. The rules in the ABM are collected from studies referenced and our exploration in the Plane World system. The ABM is discussed in detail in Section 4.2.
3. Design Process Graph (DPG): Based on ABM, the DPG describes how variables evolve from conflict situation to corresponding symptoms. The DPG is discussed in Section 4.4.

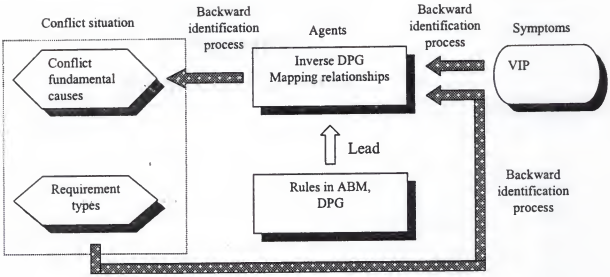
[3] From a private communication with Dr. Douglas D. Dankel, Assistant Professor of Department of Computer and Information Science and Engineering, University of Florida.

4. The inverse identification process is a backtracking through this graph, which is discussed in Section 4.5.
5. Symptoms: Symptoms, which contain the Value Indicator Patterns (VIP), are the output information associated with a conflict situation as produced by agents in the web-based CE design. Symptoms are a history of design variable values, which can be used to backtrack and determine the conflict causes. Messages between design agents may also be part of the symptoms but are not used in this study. The VIPs, that the C^2I system depends on in this work, is discussed in Section 4.3.



(a) DPG in design process

Figure 4-2: The C^2I model in (a) design process (b) inverse identification process


Conflict situation
Backward identification process
Agents
Backward identification process
Symptoms
Conflict fundamental causes
Inverse DPG Mapping relationships
VIP
Lead
Requirement types
Rules in ABM, DPG
Backward identification process

(a) DPG in inverse identification process

Figure 4-2--continued

In the design process (Figure 4-2), the conflict situation, which contains both conflict fundamental causes and requirement types, influences the agent, modeled by the ABM and DPG. Based on ABM rules, the DPG processes the input information and shows how associated variables and relations evolve as design agents seek to cooperate to achieve both local and global goals. The output design results contain the corresponding symptoms, i.e. the VIPs.

In the inverse identification process (Figure 4-2), the symptoms, i.e. the VIPs, are extracted from the design results. Also, the requirement type, stored in the public design information database that is available to the C^2I agent, serves as a key factor in the backward process determining the conflict fundamental causes. Based on the mapping between the conflict cause, requirement types and associated VIPs, the C^2I agent identifies the most likely conflict causes from the VIPs observed and requirement type available. A unique conflict cause could perhaps be obtained if a sufficiently detailed and precise model of agent behavior rules and DPGs is created and sufficient data collected. In this study, due to the limited capability of our model, the C^2I system will identify the set of possible and most likely conflict causes.

In the following sections, the four major components of the C^2I systems mentioned above, i.e. Agent Behavior Model, Value Indicator Pattern, Design Process Graph, and Inverse Identification Process will be discussed in detail.

4.2 Conflict-related Agent Behavior Model

4.2.1 A rule-oriented Agent Behavior Model (ABM)

An understanding of how the design agents behave in design, especially in a distributed CE design environment, is necessary to predict conflict symptoms for Conflict Cause Identification. The Agent Behavior Model (ABM) is used for this purpose.

In this study, the conflict-related ABM contains a set of heuristic rules collected from previously referenced studies and our exploration in the Plane World simulation. These rules describe how the agents perform and interact with each other in the distributed design environment. A general ABM would describe the agent behaviors in various aspects such as conflict-related, personality-related, etc. Here, our model is limited to the conflict related behaviors. The model has face validity in that it seems reasonable. Since this model has not been formally validated, it is considered to be a reasonable assumption.

Using the rules in natural language rather than a rigorous mathematical approach in the ABM has the following advantages,

1. Modeling human behavior often involves complex social and psychological concepts, which are generally hard to describe and analyze rigorously. Current mathematics offers little help in this task. However, rule-based modeling often works well (Hayes-Roth 1983).
2. Using natural language, the rule-based model of agent behavior is comfortable to humans and easy to understand.

4.2.2 Description of Conflict-related Agent Behavior Model

The conflict-related ABM involves two parts, i.e. design agent context and agent behavior rules.

Design Agent Context

The design agent context is a set of assumptions, describing the basic characteristics of the design agents in the design process. It contains the following items:

1. Each agent considers a complete set of significant parameters for his own part (i.e. variables and relations in his design object) that must be included in his design. However, there may be some parameters important to other agents that are not included.
2. Each agent has goals, both local and global goals, for the design of his individual part of the product, some explicit, some implicit.

3. In pursuing local goals, each agent's choice of attribute values will vary around their preferred values within a limited range, i.e. not jump around wildly.

Agent Behavior Rules

The agent behavior rules (ABR) are principles that describe how agents carry out their own part of design and coordinate with each other in a distributed design system. In this study, the following conflict-related ABRs are assumed:

1. Agents are honestly cooperative, seeking to achieve both local and global goals. (Lander, 1997b)
2. Agents will give primary attention to local goals, and secondary attention to global goals. (Ullman, 1997)
3. Agents will attempt to be cooperative and helpful when asked, but will remain local goal oriented.
4. Agents will focus on one aspect (local goal) at a time, and give significant effort toward achieving it before going on to another issue. (Ullman, 1997)
5. Agents will often delay responding to requests from others.
6. Each agent maintains internal consistency in his part of the design. There are no conflicts within each individual's part of design.
7. Agents expect other agents to think and behave like they would.
8. Upon receiving several different requests from other agents in design, the agent will focus on the latest request and tend to ignore the older ones if they

are conflicting. Thus, the latest one is considered to be most relevant to the frequently changing design situation.

In aggregation, the design agent context assumptions and the agent behavior rules listed above define the ABM used in this study.

4.3 Value Indicator Patterns

4.3.1 Pattern Recognition Approach

As Figure 4-2 shows, to identify the conflict classes and causes, the C^2I agent will track and analyze the explicit symptoms of the conflict, i.e. the output information of the conflict situation processed by agents in the forward design process. The conflict symptoms are divided into two principal types: Value Indicator Patterns and explicit messages.

Value Indicator Patterns (VIP)

The Value Indicator Pattern (VIP) is a set of numerical characteristics of a variable value history, which can be tracked and processed by the software C^2I agent toward the goal of determining the conflict class and cause. The VIP considers both current and past values of the variables, which are public to all agents involved in the design.

In this study, VIPs are used as the basis for Conflict Cause Identification and involve the following considerations,

1. Patterns of numeric variable values are easy for the software C^2I agent to retrieve and analyze. In some cases, value patterns may be the only source of design information available.

2. Value patterns are relatively easy to be interpreted and modeled using mathematical methods.

Explicit Messages

In design, agents often communicate with each other via explicit messages. Human design agents usually prefer using messages in natural language rather than the ones in a specially defined formal language, i.e. query language in Klein's system (Easterbrook 1993, Klein 1990).

Explicit messages are a potential resource for Conflict Cause Identification. However, their use has several serious drawbacks,

1. Messages are often hard to interpret by a software agent and also by human agents at times, especially the ones in natural languages. It is usually possible to set rigorous grammar standards for machine based agents. However, human agents are reluctant to comply with such grammar standards (Easterbrook 1993).
2. In some cases, there may be no messages at all. For example, some involved human design agents may feel too busy with their own work to exchange messages with others.

Thus, the C^2I system will not utilize the explicit messages but rather exclusively focus on the Value Indicator Patterns.

4.3.2 Basic Value Pattern Characteristics

In a Value Indicator Pattern, a path is formed by the time-history values of each related variable, which commonly change frequently in the design process. These may be described mathematically using an appropriate method. Interpreting the paths provides two principal types of information, i.e. the path shape and its destination. The shape demonstrates how the variable value changes with time in design. Meanwhile, from its past and current values, we can predict the curve's destination, i.e. predict its future values.

In this study, it has been found that the path's shape and destination characteristics in VIPs vary due to different combinations of conflict cause and requirement type. Therefore, the VIPs are partitioned along the two basic dimensions, i.e. Shape and Destination.

Shape

In a design process, design agents change or adjust their variables' values as they seek to achieve both their own local goals and the global design requirements. Often, the change of value of one agent's variables results in cascade adjustment of other agents' design, often with subsequent adjustment required by the original agent, forming a loop in the design process. Thus, the shape of variable paths in value patterns reveals information regarding interactions between the agents and the possible causes of the conflict.

The value patterns are grouped into the following categories of shapes:

1. Monotonic: The major trend of a variable involves steadily ascending, descending, or constant values.
2. Oscillatory: The major trend of a variable is an oscillation among several clusters or families of values.

3. Chaotic: There is no simple rule to describe the behavior of a variable. The variable values appear to be randomly distributed along the time axis.
4. Segmental: In this case, the values of a variable may be divided into several segments along the time axis. Within each segment, the variable values are clearly of one of the types listed above. However, for two adjacent segments, the values either vary greatly or are of different types.

Destination

1. Divergent: The variable values generally deviate increasingly from an acceptable value.
2. Slow convergent: The variable converges towards an acceptable value in the design process. However, the convergence speed is so slow that the variable is unlikely to achieve an acceptable value before a required deadline time.
3. Converge to an unacceptable value or stable at several values: (a) for the case of converging to an unacceptable value, the variable appears to be converging to a value which is unacceptable, i.e. not equal to an acceptable one. The predicted destination value could be either greater or less than an acceptable one. (b) For the case of stable at several values, the variable fails to converge to an acceptable value, but appears to remain stable at several values.
4. Converge to an acceptable value: The variable is predicted to converge to an acceptable value before any deadline. That is, the requirement is being satisfied by the basic design process.

Various Types of Value Indicator Pattern

The combination of the above two dimensions (i.e. shape and destination) leads to 16 types of Value Indicator Patterns (VIPs), shown in Table 4-1. Examples of these distinctive VIP types involving two agents are shown graphically in Figure 4-3 to Figure 4-6.

Table 4-1: Abbreviations used to identify Value Indicator Pattern

Destination / Shape	Divergent (DV)	Slow convergent (SC)	Converge to an unacceptable value/stable at several values (CUS/SSV)	Converge to an acceptable value (CAV)
Monotonic (MO)	MO-DV	MO-SC	MO-CUV/SSV	MO-CAV
Oscillatory (OS)	OS-DV	OS-SC	OS-CUV/SSV	OS-CAV
Chaotic (CH)	CH-DV	CH-SC	CH-CUV/SSV	CH-CAV
Segmental (SG)	SG-DV	SG-SC	SG-CUV/SSV	SG-CAV

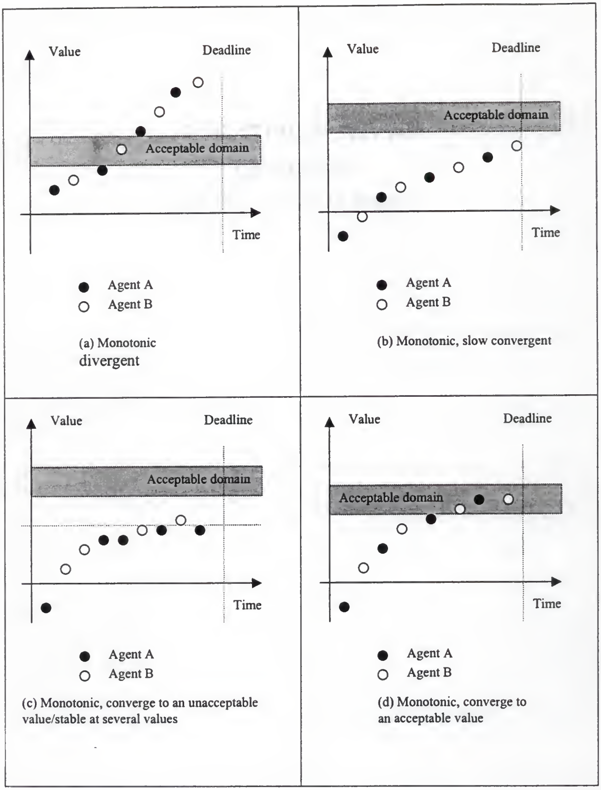


Figure 4-3: Various monotonic value patterns

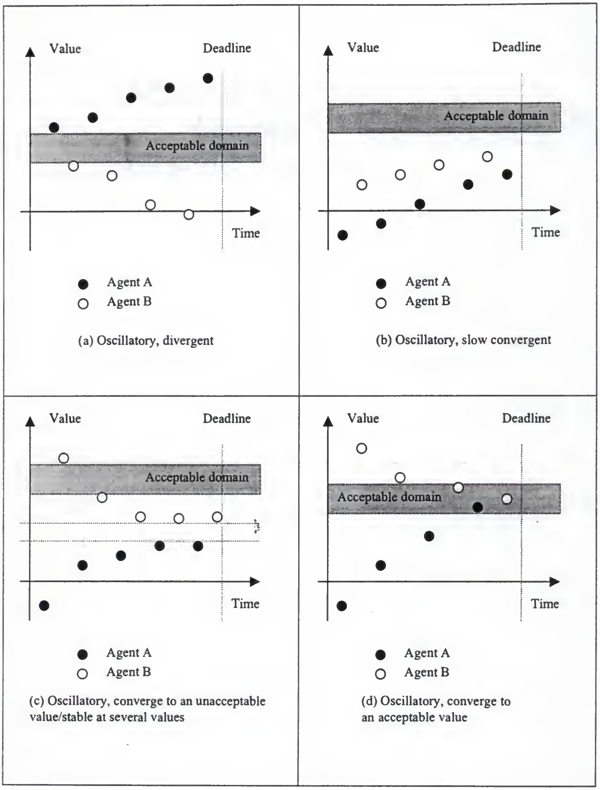


Figure 4-4: Various oscillatory value patterns

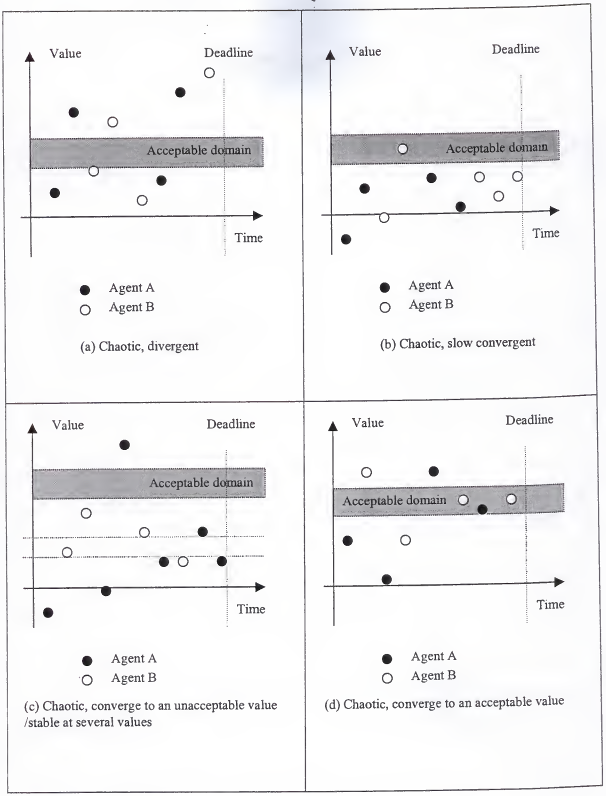


Figure 4-5: Various chaotic value patterns

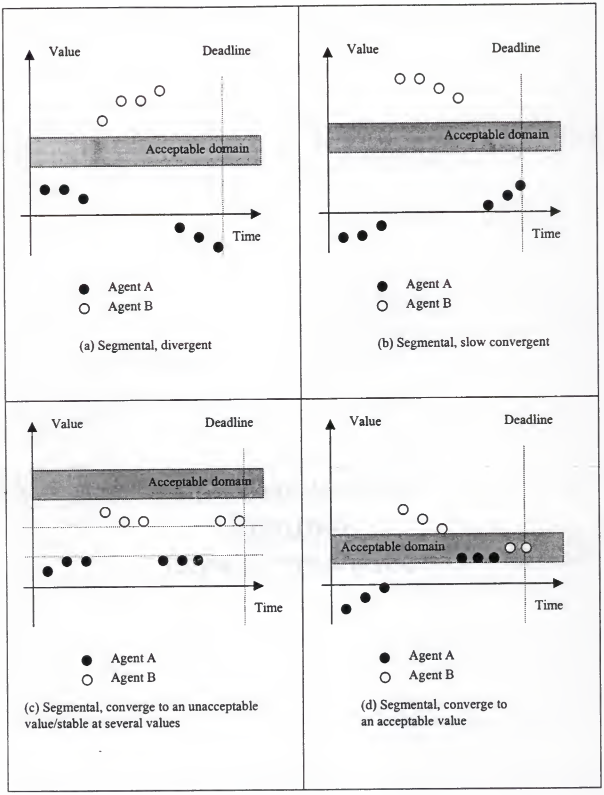


Figure 4-6: Various segmental value patterns

4.3.3 Real World Conflict-related Value Indicator Patterns

In this work, Value Indicator Patterns are sought which form logical cohesive set that reflects real world design process behavior, and that can be reliably recognized by a software agent. The following examples illustrate that the VIPs chosen can and do arise in real world design efforts.

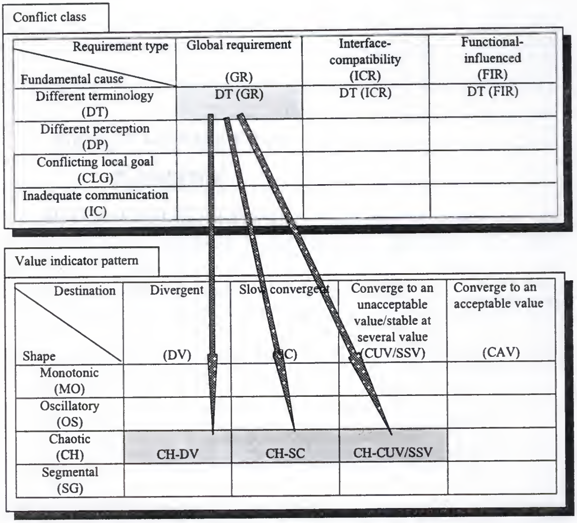
As discussed in the above sections, the C^2I model assumes different combinations of conflict fundamental causes and requirement types commonly lead to different Value Indicator Patterns (VIPs). This section provides limited confirmation of this assumption. Here, several examples of different conflict classes and corresponding VIPs collected in literature referenced and Plane World simulations are discussed. Focus is on description of these VIPs. Details about how these VIPs are produced by results from the conflict situations will be discussed in Section 4.4. For convenience, the abbreviations used in referring to conflict classes are shown in Table 4-2.

Table 4-2: Abbreviation of conflict classes

Requirement type / Conflict cause	Global (GR)	Interface-compatibility (ICR)	Functional-influenced (FIR)
Different terminology (DT)	DT (GR)	DT (ICR)	DT (FIR)
Different perception (DP)	DP (GR)	DP (ICR)	DP (FIR)
Conflicting local goal (CLG)	CLG (GR)	CLG (ICR)	CLG (FIR)
Inadequate communication (IC)	IC (GR)	IC (ICR)	IC (FIR)

Example 1: Different Terminology, Global Requirement

Figure 4-7 shows the mapping from the conflict class "different terminology, global requirement" to the associated VIPs.



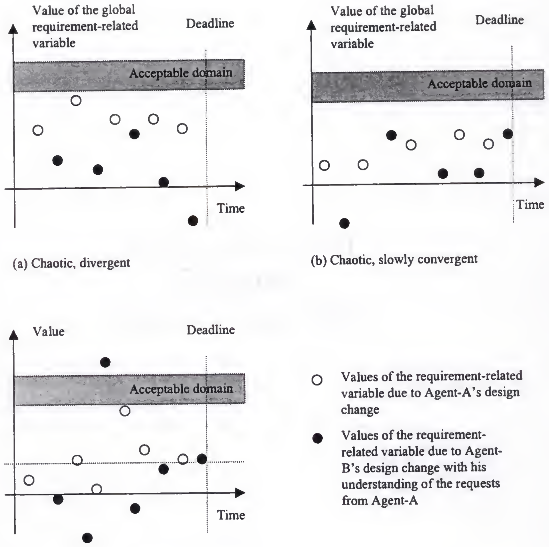
Conflict class

Requirement type / Fundamental cause	Global requirement (GR)	Interface-compatibility (ICR)	Functional-influenced (FIR)
Different terminology (DT)	DT (GR)	DT (ICR)	DT (FIR)
Different perception (DP)			
Conflicting local goal (CLG)			
Inadequate communication (IC)			

Value indicator pattern

Destination / Shape	Divergent (DV)	Slow convergent (SC)	Converge to an unacceptable value/stable at several value (CUV/SSV)	Converge to an acceptable value (CAV)
Monotonic (MO)				
Oscillatory (OS)				
Chaotic (CH)	CH-DV	CH-SC	CH-CUV/SSV	
Segmental (SG)				

Figure 4-7: Mapping from the conflict class "different terminology, global requirement" to associated VIPs



(a) Chaotic, divergent

(b) Chaotic, slowly convergent

(c) Chaotic, converge to an unacceptable value or stable at several values

Figure 4-8: An example of VIPs expected to result from the conflict class "different terminology, global requirement"

Consider the example of cargo capacity in aircraft design discussed in Section 3.4.2; two design agents responsible for the cargo and structure modules have different understanding of the term "cargo capacity". For instance, the cargo agent considers "cargo capacity" as the minimum width and height of the cargo cabin, while the structure agent interprets it as the maximum cargo weight. Meanwhile, there is a global requirement on the cargo performance measure, i.e. the minimum height and width of the cargo cabin.

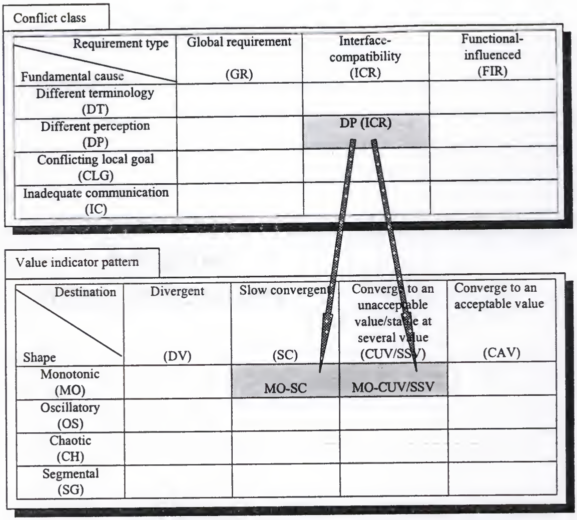
Different terminology commonly leads to misunderstanding between the two agents and conflicts in their designs. Thus, the behavior of each agent is irrational to the other and there appears to be no correlation between results. The VIPs which might result are illustrated in Figure 4-8. In the figure, the shape of design result (i.e. the global requirement-related variable) is chaotic, i.e. there is no simple rule to describe its behavior. Meanwhile, there are three possible destinations of the design result, i.e. (a) divergent, (b) slowly convergent, or (c) converge to an unacceptable value/stable at several values, as described in Section 4.3.2.

Example 2: Different Perception, Interface-Compatibility Requirement

The mapping from the conflict class "different perception, interface-compatibility requirement" to the associated VIPs is shown in Figure 4-9.

To illustrate the VIPs of this class, consider the example of interface design between the propulsion and structure modules discussed in Section 3.4.1. In the example, module variables involved in the interface-compatibility requirement are balanced, in that they are not functionally dependent on each other but are assigned by their module agents. They are, of course, required to be compatible in values. The two agents responsible for these two modules have different perceptions regarding desirable hole-positions of attachment components, i.e. bolts and nuts. For instance, the propulsion agent might prefer wide spacing between bolts for saving the space for other links (i.e. fuel, electric etc.), while the structure agent might prefer smaller spacing for even distribution of the load. Each agent works on his module design primarily based on his own perception and is reluctant to change, perhaps hoping the other agent will change. Thus, the requirement that the hole-positions in the two modules should match will not be satisfied, a conflict in design.

Possible VIPs resulting from this conflict class are illustrated in Figure 4-10. The path shape of the requirement-related variable value (i.e. difference between the two designs) is expected to be monotonic. The destination could be one of two types, i.e. slowly convergent and converge to an unacceptable value/stable at several values, as described in Section 4.3.2.



Conflict class

Requirement type / Fundamental cause	Global requirement (GR)	Interface-compatibility (ICR)	Functional-influenced (FIR)
Different terminology (DT)			
Different perception (DP)		DP (ICR)	
Conflicting local goal (CLG)			
Inadequate communication (IC)			

Value indicator pattern

Destination / Shape	Divergent (DV)	Slow convergent (SC)	Converge to an unacceptable value/stable at several value (CUV/SSV)	Converge to an acceptable value (CAV)
Monotonic (MO)		MO-SC	MO-CUV/SSV	
Oscillatory (OS)				
Chaotic (CH)				
Segmental (SG)				

Figure 4-9: Mapping from the conflict class "different perception, interface requirement" to associated VIPs

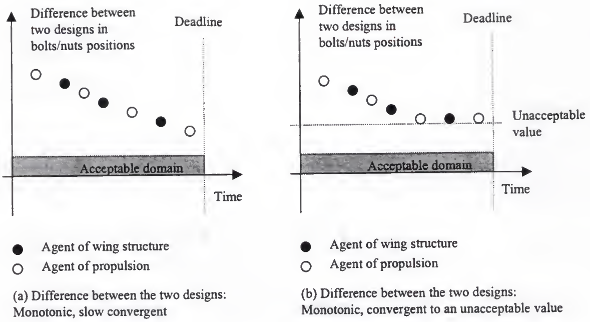


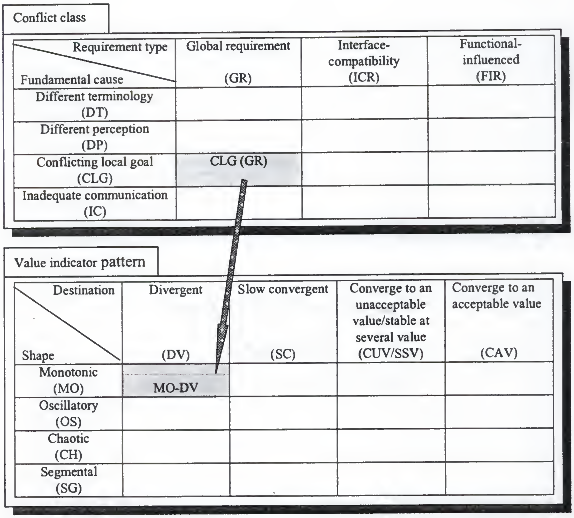
Figure 4-10: An example of VIPs expected to result from the conflict class "different point of view, interface-compatibility requirement"

Example 3: Conflicting Local Goals, Global Requirement

Figure 4-11 shows the mapping from the conflict class "conflicting local goals, global requirement" to the associated Value Indicator Pattern.

To illustrate the VIPs of this class, consider the example of tank design discussed in Section 3.4.2, the areas of gunnery, vehicle structure, armor, and engine are functionally connected to the global requirements of overall weight and performance measures (i.e. firepower, mobility, and protection).

In the design, the design agents responsible of the vehicle structure, armor, and engine modules have conflicts in achieving their local goals. One agent's design change results in cascade changes in other agents' designs, resulting in a positive feedback loop and steady increase of the tank overall weight. Conflict becomes clear when the weight goes beyond an acceptable value.



Conflict class

Requirement type / Fundamental cause	Global requirement (GR)	Interface-compatibility (ICR)	Functional-influenced (FIR)
Different terminology (DT)			
Different perception (DP)			
Conflicting local goal (CLG)	CLG (GR)		
Inadequate communication (IC)			

Value indicator pattern

Destination / Shape	Divergent (DV)	Slow convergent (SC)	Converge to an unacceptable value/stable at several value (CUV/SSV)	Converge to an acceptable value (CAV)
Monotonic (MO)	MO-DV			
Oscillatory (OS)				
Chaotic (CH)				
Segmental (SG)				

Figure 4-11: Mapping from the conflict class "conflicting local goal, global requirement" to the associated VIPs

The associated VIP is shown in Figure 4-12. Here, the path shape of the requirement-related value (i.e. weight of the tank) is monotonic. Its destination is divergent, i.e. steadily increasing beyond the acceptable value due to the positive feedback loop caused by conflicting local goals among design agents.

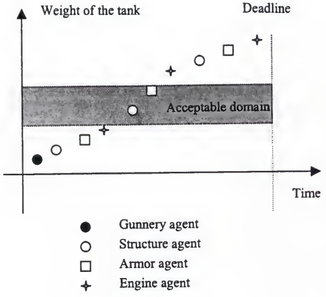


Figure 4-12: An example of the VIP expected to result from the conflict class "conflicting local goals, global requirement"

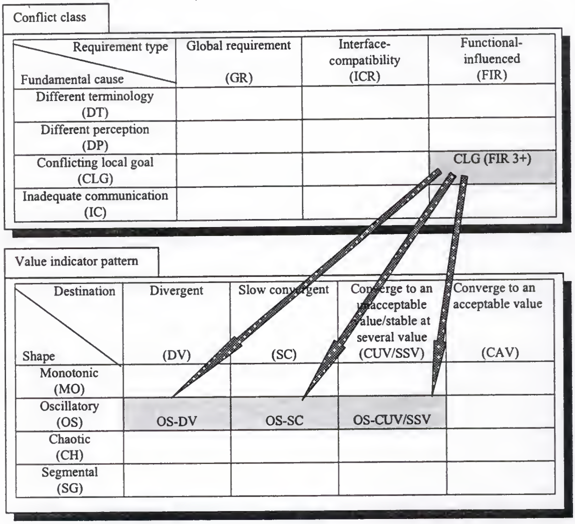
Example 4: Conflicting Local Goals, Functional-Influenced Requirement

Figure 4-13[4] shows the mapping from the conflict class "conflicting local goals, functional-influenced requirement" to the associated VIPs. For example, the aircraft design is divided into several parts, i.e. control, propulsion and structure assigned to different teams. Assuming the orientation of the propulsion exhaust is aftwards, values of some variables in both control and tail structure parts are influenced by the location and orientation of the propulsion unit.

[4] In the figure, "FIR 3+" means functional-influenced requirement type with three or more agents involved.

For the control part, the yaw-controllability (Anderson 1989) when one of the two propulsion units shuts down is influenced by the propulsion unit's location and orientation.

For tail structure, the vibration and heat loading is also influenced by the propulsion unit's location and orientation. The situation is illustrated in Figure 4-14.



Conflict class

Requirement type / Fundamental cause	Global requirement (GR)	Interface-compatibility (ICR)	Functional-influenced (FIR)
Different terminology (DT)			
Different perception (DP)			
Conflicting local goal (CLG)			CLG (FIR 3+)
Inadequate communication (IC)			

Value indicator pattern

Destination / Shape	Divergent (DV)	Slow convergent (SC)	Converge to an unacceptable value/stable at several value (CUV/SSV)	Converge to an acceptable value (CAV)
Monotonic (MO)				
Oscillatory (OS)	OS-DV	OS-SC	OS-CUV/SSV	
Chaotic (CH)				
Segmental (SG)				

Figure 4-13: Mapping from the conflict class "conflicting local goal, functional-influenced requirement (3 or more agents involved)" to associated VIPs

Here, the local goals of the agents responsible for the control and tail structure parts are conflicting with respect to values of the propulsion unit's location. The control agent wants the propulsion units to be placed close to the fuselage for better yaw controllability when one of the propulsion units is shut down. In contrast, the structure agent responsible for the tail structure wants the propulsion units to be located away from the fuselage so the tail can be cleared from the propulsion units' exhaust.

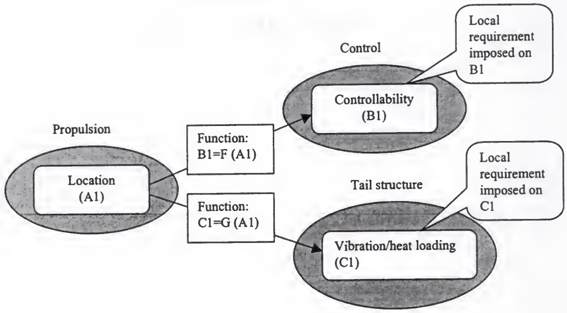


Figure 4-14: An example of a functional-influence requirement

Because of the conflicting local goals, the agents responsible for the control and tail-structure modules send opposite requests to the propulsion agent about the propulsion's location, which leads to conflict in design. Since the ABM suggests that agents tend to respond to the most recent request, an oscillatory shape is anticipated.

Possible VIPs resulted from conflicting local goals are illustrated in Figure 4-15. In the VIPs, the shape of the requirement-related variables (i.e. the distance from the between the propulsion and the tail for the structure part and the distance between the propulsion unit and the fuselage for the control part) tends be oscillatory. The destinations could be (a) divergent, (b) slowly convergent, or (c) converge to an unacceptable value / stable at several values, as described in Section 4.3.2.

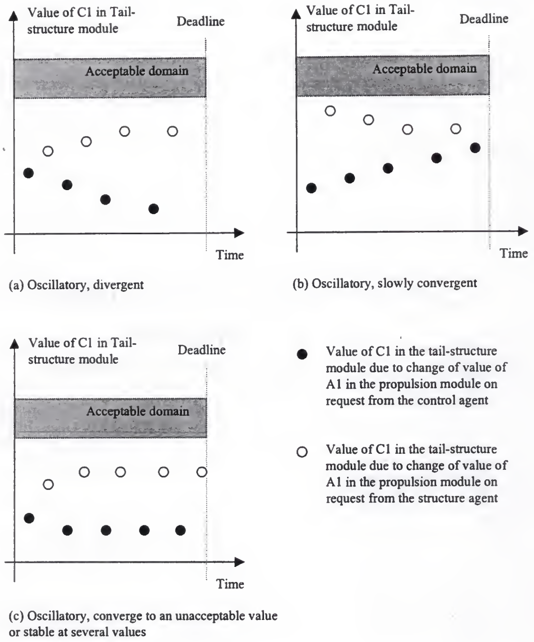
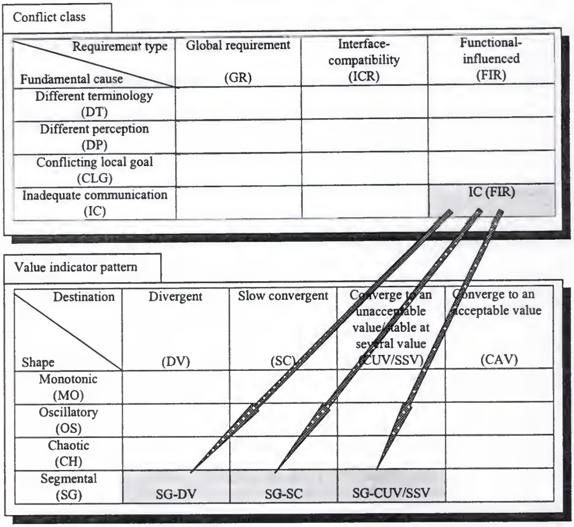


Figure 4-15: An example of VIPs expected to result from the conflict class "conflicting local goal, functional-influenced requirement (3 or more agents involved)"[5]

[5] The VIPs of the requirement-related variable (i.e. B1 in the control module) are similar and omitted here.

Example 5: Inadequate Communication, Functional-Influenced Requirement

Figure 4-16 shows the mapping between the conflict class "inadequate communication, functional-influenced requirement" and the associated VIPs.



Conflict class

Requirement type / Fundamental cause	Global requirement (GR)	Interface-compatibility (ICR)	Functional-influenced (FIR)
Different terminology (DT)			
Different perception (DP)			
Conflicting local goal (CLG)			
Inadequate communication (IC)			IC (FIR)

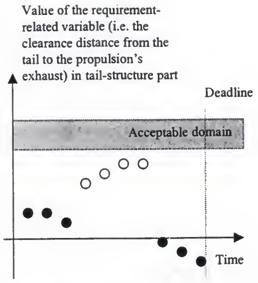
Value indicator pattern

Destination / Shape	Divergent (DV)	Slow convergent (SC)	Converge to an unacceptable value/stable at several value (CUV/SSV)	Converge to an acceptable value (CAV)
Monotonic (MO)				
Oscillatory (OS)				
Chaotic (CH)				
Segmental (SG)	SG-DV	SG-SC	SG-CUV/SSV	

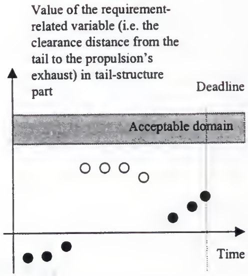
Figure 4-16: Mapping from the conflict class "inadequate communication, functional-influenced requirement" to the associated VIPs

To illustrate the VIPs of this class, consider the example of aircraft design discussed in Section 3.4.2, which is decomposed into multiple teams responsible for different modules of the aircraft. Meanwhile, there is a functional-influenced requirement that the tail must be cleared from the exhaust of the propulsion units. Otherwise, the exhaust will cause high loading of heat, noise and vibration on the tail, leading to unacceptable damage to its structure. In design, some agents (i.e. the propulsion agent) may totally focus on their own modules and ignore the request messages or design changes from other design agents (i.e. the structure agent responsible for the tail), these may be inadequate communication between design agents leading to conflicts in the design. Thus, there are extended periods of inadequate communication followed by response.

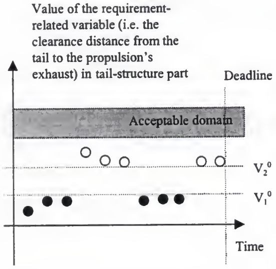
Possible VIPs are illustrated in Figure 4-17. Here, the path shape of the value of the requirement-related variable (i.e. the clearance distance from the tail to the effect zone of the propulsions) is segmental. That is, the history values are divided into several segments. Within each segment, the values are relatively constant or converging toward a constant value. However, the values between two adjacent segments are significantly different. The destination of the requirement-related variable could be one of three types; i.e. (a) divergent, (b) slowly convergent, or (c) converge to an unacceptable value/stable at several values, as described in Section 4.3.2.



(a) Segmental, divergent



(b) Segmental, slow convergent



(c) Segmental, converge to an unacceptable value or stable at several values

● Value of the requirement-related variable (i.e. the clearance distance from the tail to the propulsion's exhaust) when the propulsion agent focuses on his part of design and tends to ignore the tail structure agent's requests

○ Value of the requirement-related variable (i.e. the clearance distance from the tail to the propulsion's exhaust) after the propulsion agent considers the tail structure agent's request and adjust his part of design

Figure 4-17: An example of VIPs expected to result from the conflict class "Inadequate communication, functional-influenced requirement"

4.4 Design Process Graph in the Forward Design Process

Based on the ABM, Design Process Graphs (DPG) are developed to describe how the conflict fundamental causes lead to the specific Value Indicator Patterns (VIP) and the evolution of variables and relations in this procedure (Figure 4-3a). Thus, the mapping between conflict cause/requirement type and associated VIPs is identified and studied. In the inverse identification process, such mapping is inverted to determine the conflict fundamental causes from the VIPs observed and the requirement types available.

In the following section, the DPGs and their derivation of VIPs is presented. The inverse identification in which possible conflict class and cause are determined is presented in Section 4.5.

4.4.1 Structure of the Design Process Graph

The DPG contains the following elements:

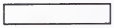
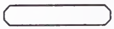
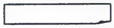
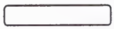
1. Conflict cause: The fundamental causes of conflict. These are the principal types of agent disagreements leading to conflicts in collaborative engineering designs. In this study, seven fundamental causes are identified, defined, and studied.
2. Requirement type: Requirement type, in combination with conflict cause, plays a key role in determining the conflict symptoms. In this study, three principal types of requirements are identified, (a) global, (b) interface-compatibility, and (c) functional-influenced.

A further possible subdivision of requirement type is to the cases of requirement relation type[6], which may prove useful in the future but is not used here.

3. Agent Behavior Rules (ABR): Rules in the ABM serve as a guideline in construction of the process graphs, describing how the design agents are predicted to behave and how the variables with relevance to several modules are expected to evolve in the design process.
4. Variable: The variables mentioned here include both local variables in individual modules and global variables related to global requirements.
5. Alternative patterns: The alternative patterns of a variable represent the patterns, which may result from the choices made by design agents.

The symbols for the above elements used in figures of the DPG applied in the C^2I model are shown in Table 4-3.

Table 4-3: Elements in the DPG

ELEMENTS IN DPG	SYMBOLS
Conflict fundamental cause	
Design agent	
Agent Behavior Rule (ABR)	
Variable value	

[6] There are many different types of relations in the requirement such as (a) $V < V_0$, (b) $V > V_0$, (c) $V \in [V_1, V_2]$, and (d) $V \notin [V_1, V_2]$.

Alternative pattern	▭
Requirement satisfied or not	⬭
Specification of values	⟶
Information, or messages	┈▶
ABM rules determine the agent behavior	-·-·-▶

As discussed in Section 3.3.1, the conflicts are divided into different classes, represented by cells in the matrix illustrated in Figure 3-1. Each class (i.e. Class C_{ij}) is a combination of fundamental cause and requirement type (indicated by i and j). In this work, DPGs are introduced to describe how variables evolve in these different conflict classes.

Concurrent Engineering (CE) design commonly involves loops in design process. Figure 4-18 illustrates the interaction between agents and the resulting design loops in a design situation involving in a particular conflict class (i.e. Class C_{ij}). In the design, Agent-A is responsible for variable VAR-A1 of module A, and Agent-B is responsible for VAR-B1 of module B. A requirement is imposed on a variable, i.e. VAR-R, which is determined by values of variables VAR-A1 and VAR-B1 through functions. According to the ABM rules, design agents seek to achieve both local and global goals. That is, they specify their variable values based on their own local goal evaluation and feedback from the public information (i.e. value patterns of the requirement-related variable VAR-R) and other agents (i.e. explicit messages). Therefore, Agent-A and Agent-B often alternate in adjusting their variable values in order to cope with the other party's change in design, resulting in a series of loops as Figure 4-19 shows.

The important role of loops is a major difference between the Design Process Graph (DPG) and the decision trees commonly used in expert systems, which often do not involve loops (Raiffa 1970, Buchanan 1984).

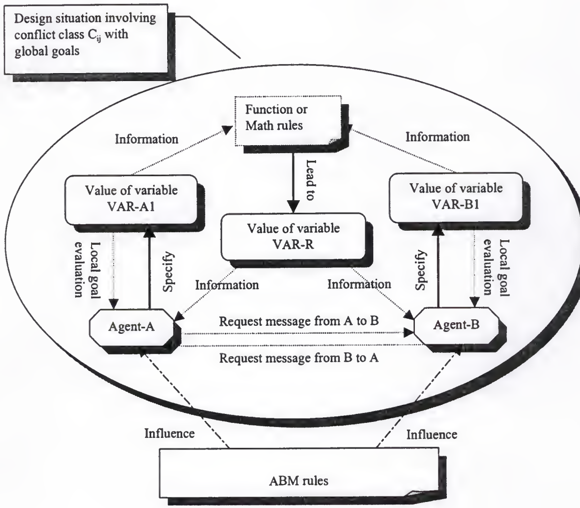


Figure 4-18: Model of the interaction between agents in CE design

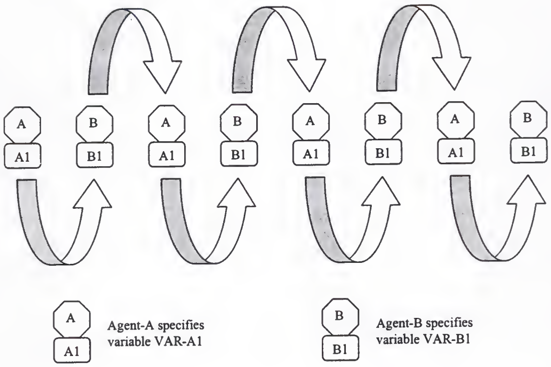

A
A1
B
B1
Agent-A specifies variable VAR-A1
Agent-B specifies variable VAR-B1

Figure 4-19: Results from the design loop

Figure 4-20 illustrates the DPG of Case C_{ij}. Here, one agent (i.e. Agent-A) adjusts the value of his module variable (i.e. VAR-A1) to achieve his local goal. The change of Agent-A's variable VAR-A1 may affect other agents' variables, which are functional-related (i.e. Agent-B's variable VAR-B1). Thus, based on the agent behavior rules, Agent-B is expected to adjust the values of VAR-B1 to achieve his local goals and the requirements. Often, Agent-A has to adjust VAR-A1 due to the change of VAR-B1 by Agent-B. Thus, the interaction among agents forms loops in the design process. In consequence, the associated requirement may be affected, producing one of several VIPs (i.e. VIP1 and VIP2). The requirement may or may not be satisfied, indicated by the DPG.

In decision trees, not all paths (i.e. series of decisions) have to lead to desirable results (Raiffa 1970). Similarly, the paths in DPGs do not always lead to a violation of the requirement. If the requirement is violated, the information about how variables evolve along this path is composed to a Value Indicator Pattern (VIP) of this specific case. Here, only those paths that lead to requirement violation are considered in the DPGs in this study.

4.4.2 Use of the Design Process Graph in the Forward Design Process

In this section, analyses are described which show how the Design Process Graph (DPG), implicitly defined by ABM rules, leads to the change of designing state (i.e. variables, relations and requirements) in various cases of conflict fundamental causes and requirement types and leads to specific Value Indicator Patterns. Results are collected in Table 4-4.

Analysis 1: Different Terminology, Global Requirement

As introduced in Section 4.3.3, the mapping from the conflict class of different terminology with global requirement to the associated VIPs is shown in Figure 4-7.

Conflicts caused by different terminology may involve any one of the three requirement-types, i.e. global, interface-compatibility, and functional-influenced. Here, the situation involving global requirement (Class C_{11}) and its DPG is analyzed. This will be done using a specific example of cargo capacity in aircraft design (introduced in Section 4.3.3), where two agents are responsible for designs of cargo and structure modules (i.e. Agent-A for the cargo module and Agent-B for the structure module). The two agents have different meaning when they use the same term "cargo capacity".

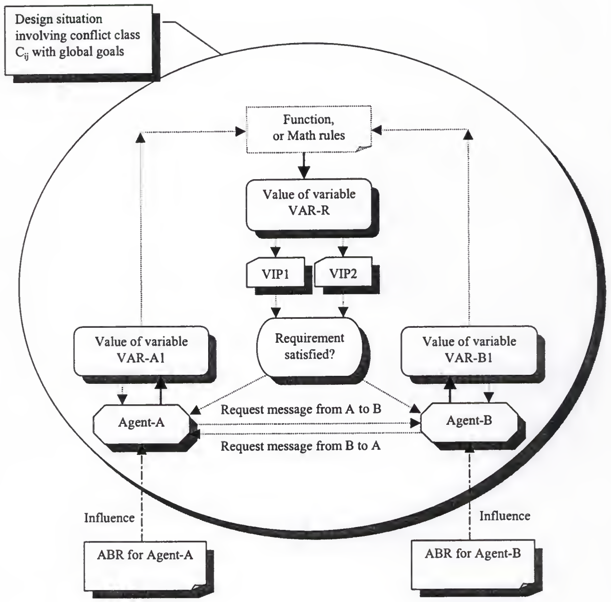


Figure 4-20: The DPG of conflict class C_{ij} in the forward design process

In design, Agent-A sends a message to Agent-B asking him to increase cargo capacity (meaning the minimum width and height) of the cargo space. Due to different interpretation, instead of making more space available for the cargo, Agent-B strengthens the structure of the cargo space based on his understanding of the term "cargo capacity" (i.e. interpreted as meaning the maximum cargo weight). In other word, Agent-B moves in a different direction than what is expected by Agent-A. In consequence, the related variables, the cargo space's minimum width and height, do not change in any anticipated way and the requirements are not satisfied. To Agent-A, Agent-B's misunderstanding and subsequent action is irrational and unpredictable. So is the design result. The shape of the requirement-related variable values, i.e. the minimum width and height of cargo cabin, tends to be chaotic. Its destination could be divergent, slow convergent, or converge to an unacceptable value/stable at several values. The process is illustrated in Figure 4-21. The resulting VIPs are expected to be similar to those shown in Figure 4.8.

Analysis 2: Different Terminology, Interface-Compatibility Requirement

To analyze the design process for the conflict class "different terminology, interface-compatibility requirement", assume two agents (i.e. Agent-A and Agent-B) are responsible for the designs of the interface between two modules (i.e. Module A and B). There is an interface-compatibility requirement that the two modules' interface design should be compatible, i.e. the difference between values of variable VAR-A1 in Module A and VAR-B1 in Module B should be within an acceptable small range:

$|VAR\text{-}A1 - VAR\text{-}B1| < \varepsilon$

The two agents have different meaning when they use the term "$TERM_X$", for instance, Agent-A considers $TERM_X$ as a variable VAR-A1 while Agent-B interprets it as another variable VAR-B2 rather than VAR-B1, the counterpart of VAR-A1.

In the design, Agent-A asks Agent-B via messages to change the value of $TERM_X$ (meaning VAR-B1) to match his design of VAR-A1. Based on different understanding of the term "TERM$_X$" (i.e. interpreted as meaning VAR-B2), Agent-B tends to change the value of VAR-B2, which is not what Agent-A expects. Thus, the requirement on the difference between values of VAR-A1 and VAR-B1 will not be satisfied.

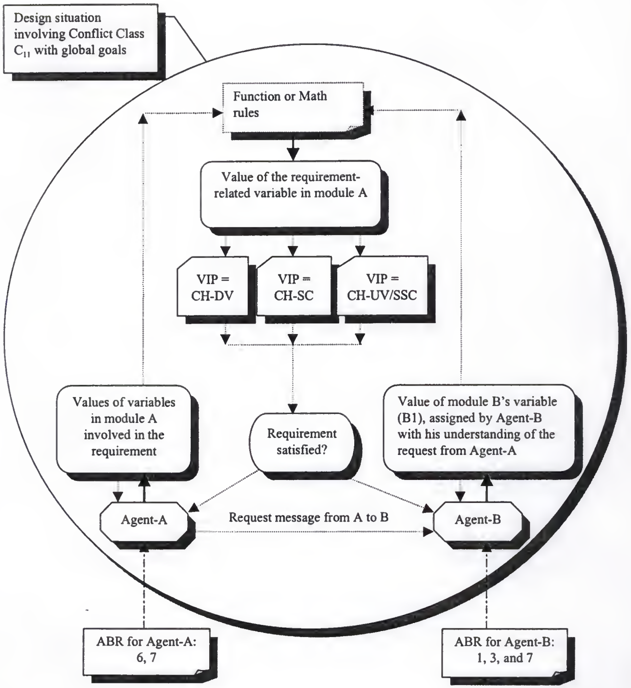

Design situation involving Conflict Class C$_{11}$ with global goals
Function or Math rules
Value of the requirement-related variable in module A
VIP = CH-DV
VIP = CH-SC
VIP = CH-UV/SSC
Values of variables in module A involved in the requirement
Requirement satisfied?
Value of module B's variable (B1), assigned by Agent-B with his understanding of the request from Agent-A
Agent-A
Request message from A to B
Agent-B
ABR for Agent-A: 6, 7
ABR for Agent-B: 1, 3, and 7

Figure 4-21: DPG of the conflicts class "different terminology, global requirement"

Since Agent-B's misinterpretation and subsequent cannot be described in a simple rule, the shape of the design result tends to be chaotic. Its destination could be divergent, slow convergent, or converge to an unacceptable value/stable at several values.

Figure 4-22 illustrates the process. The resulting VIPs are also expected to be similar to those shown in Figure 4-8.

Analysis 3: Different Terminology, Functional-Influenced Requirement

In this case, two design agents (i.e. Agent-A and Agent-B) are responsible for designs of two modules (i.e. Module A and Module B) where a functional-influenced requirement is imposed on a variable in Module A (i.e. VAR-A1) whose value is controlled by the value of another module's variable (i.e. VAR-B1). These two agents use different terms on VAR-B1. For example, Agent-A uses T_A for VAR-B1, while T_A means another variable to Agent-B.

Thus, in the design, when Agent-A sends a message asking Agent-B to adjust VAR-B1 to his desired value, Agent-B tends to change some other variable's value due to the misunderstanding of Agent-A's terms. Similar to the situation in Analysis 1 and 2, Agent-B's misunderstanding and his design is unpredictable, the path of design result is chaotic. Its destination could be divergent, slow convergent, or converge to an unacceptable value/stable at several values.

The process is illustrated in Figure 4-23. The resulting VIPs are also expected to be similar to those shown in Figure 4-8.

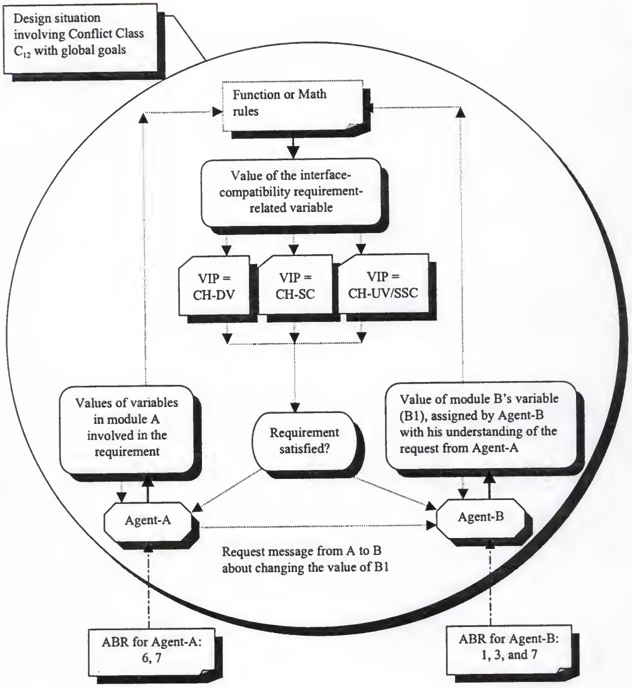

Design situation involving Conflict Class C_{12} with global goals
Function or Math rules
Value of the interface-compatibility requirement-related variable
VIP = CH-DV
VIP = CH-SC
VIP = CH-UV/SSC
Values of variables in module A involved in the requirement
Requirement satisfied?
Value of module B's variable (B1), assigned by Agent-B with his understanding of the request from Agent-A
Agent-A
Agent-B
Request message from A to B about changing the value of B1
ABR for Agent-A: 6, 7
ABR for Agent-B: 1, 3, and 7

Figure 4-22: DPG of the conflict class "different terminology, interface-compatibility requirement"

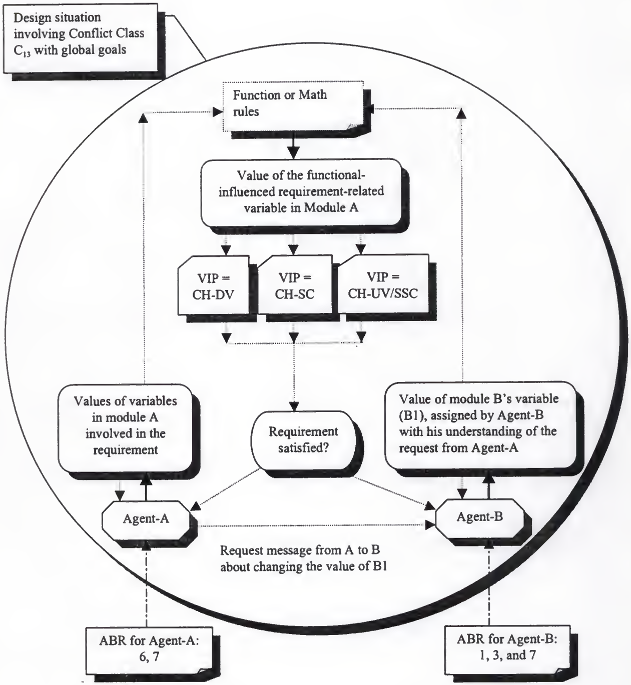

Design situation involving Conflict Class C13 with global goals
Function or Math rules
Value of the functional-influenced requirement-related variable in Module A
VIP = CH-DV
VIP = CH-SC
VIP = CH-UV/SSC
Values of variables in module A involved in the requirement
Requirement satisfied?
Value of module B's variable (B1), assigned by Agent-B with his understanding of the request from Agent-A
Agent-A
Agent-B
Request message from A to B about changing the value of B1
ABR for Agent-A: 6, 7
ABR for Agent-B: 1, 3, and 7

Figure 4-23: DPG of the conflict class "different terminology, functional-influenced requirement"

Analysis 4: Different Perception, Global Requirement

In this conflict class, two or more agents (i.e. Agent-A, B, etc.) responsible for different modules are involved in a global requirement, which is imposed on a global variable (VAR-GR). In the design, the agents may have different perceptions of the module designs (i.e. module variables) related to the global requirement-related variable. According to ABM rules, each agent design his module based on his own local perception, which may not satisfy the global requirement. To achieve the global requirement, the design agents seek to be cooperative and narrow the gap between their different perceptions. However, because the agents are primarily local perception oriented and reluctant to budge from their initial designs, the converging process tends to be too slow to reach the acceptable value of VAR-GR or even becomes stuck at an unacceptable value. Therefore, the shape of the design result (i.e. VAR-GR) tends to be monotonic. Its destination could be either slow convergent or converge to an unacceptable value/stable at several values.

The process is shown in Figure 4-24. The resulting VIPs could be similar to those shown in Figure 4-10.

Analysis 5: Different Perception, Interface-Compatibility Requirement

The mapping, to be analyzed, from the conflict class "different perception, interface-compatibility requirement" to the associated VIPs, is shown in Figure 4-9.

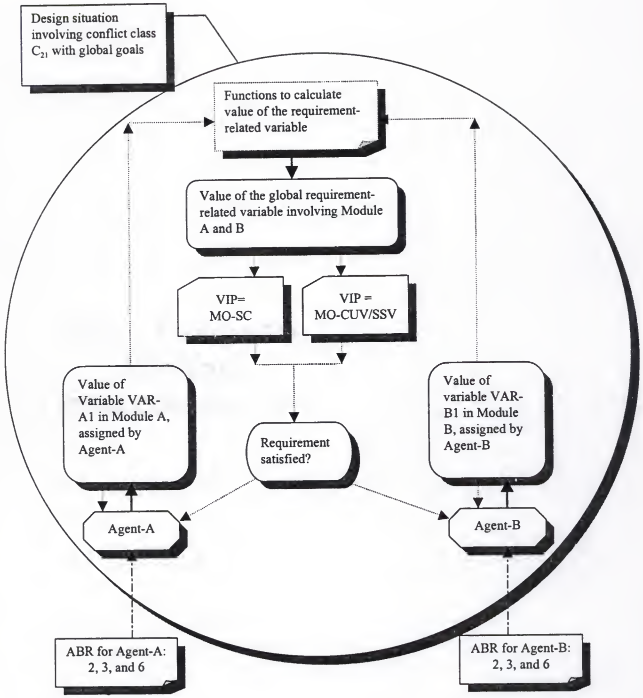

Design situation involving conflict class C_{21} with global goals
Functions to calculate value of the requirement-related variable
Value of the global requirement-related variable involving Module A and B
VIP= MO-SC
VIP = MO-CUV/SSV
Value of Variable VAR-A1 in Module A, assigned by Agent-A
Requirement satisfied?
Value of variable VAR-B1 in Module B, assigned by Agent-B
Agent-A
Agent-B
ABR for Agent-A: 2, 3, and 6
ABR for Agent-B: 2, 3, and 6

Figure 4-24: DPG of the conflict class "different perception, global requirement"

To analyze a DPG of this class, consider the example of attachment-interface design between the propulsion and structure modules in airplane design (see also Section 3.4.3) where the two agents working on the propulsion and structure modules have different perceptions on the hole-positions of bolts and nuts based on their different background and experience. There is, however, the requirement that the two module-designs at the interface should be compatible; i.e. the hole-positions of the two modules should be same ($V_{\text{Difference}} < \varepsilon_0$).

Based on the ABM rules, each agent chooses the hole-position for his module primarily based on his own perception, which is expected to lead to incompatibility between the two designs at the interface. For instance, Agent-A's design of the hole-position converges to one value while Agent-B's design converges to another value. Meanwhile, the agents try to be cooperative, according to the ABM rules. The difference between the two designs would likely be narrowed monotonically as the agents try to close the gap between their perceptions in the design process. Often loops occur as agents adjust the values of their variables to accommodate the others' designs. However, the narrowing of the gap between these two interface designs is often slow because the agents are primarily concerned with their own local goals and local perceptions. The design process may become stuck when the two agents are reluctant to budge from their own positions. The difference between the two designs then converges to an unacceptable value, i.e. a value greater than the required one. The process is illustrated in Figure 4-25. The resulting patterns are expected to be similar to those shown in Figure 4-10.

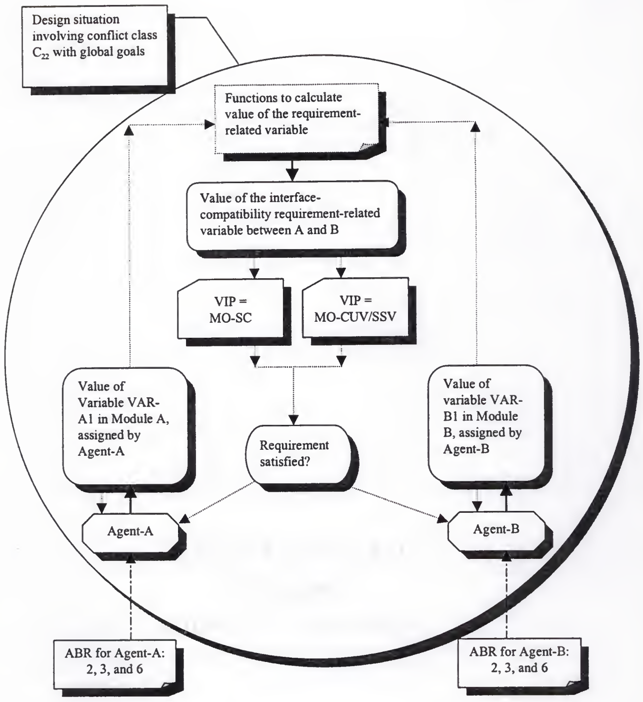

Design situation involving conflict class C_{22} with global goals
Functions to calculate value of the requirement-related variable
Value of the interface-compatibility requirement-related variable between A and B
VIP = MO-SC
VIP = MO-CUV/SSV
Value of Variable VAR-A1 in Module A, assigned by Agent-A
Requirement satisfied?
Value of variable VAR-B1 in Module B, assigned by Agent-B
Agent-A
Agent-B
ABR for Agent-A: 2, 3, and 6
ABR for Agent-B: 2, 3, and 6

Figure 4-25: DPG of the conflict class "different perception, interface-compatibility requirement"

Analysis 6: Different Perception, Functional-Influenced Requirement

This conflict class involves two cases: the requirement involving two design agents and the one involving three or more agents.

In the first case, assume two agents (i.e. Agent-A and Agent-B) are responsible for Module A and B. There is a functional-influenced requirement imposed on a variable in Module A (i.e. VAR-A1) whose value is controlled by a variable in Module B (i.e. VAR-B1). The two agents have different perceptions on the desirable value of VAR-B1. For instance, Agent-A prefers a bigger value while Agent-B prefers a smaller value.

According to the ABM rules, in the design, each agent specifies the values of variables in his module based on his own perception. Thus, VAR-B1 specified by Agent-B tends to be of small value. Based on different perception, Agent-A sends a message to Agent-B asking him to change VAR-B1 to a larger value. Seeking to be cooperative, Agent-B tries to increase the value of VAR-B1 if it does not affect his own local goals. However, the agents are primarily local perception and local goal oriented. The converging process tends to be very slow or even stuck to an unacceptable value. Therefore, the path shape of the history values of the requirement-related variable (i.e. VAR-A1) tends to be monotonic. Its destination could be either slowly convergent or converging to an unacceptable value/stable at several values.

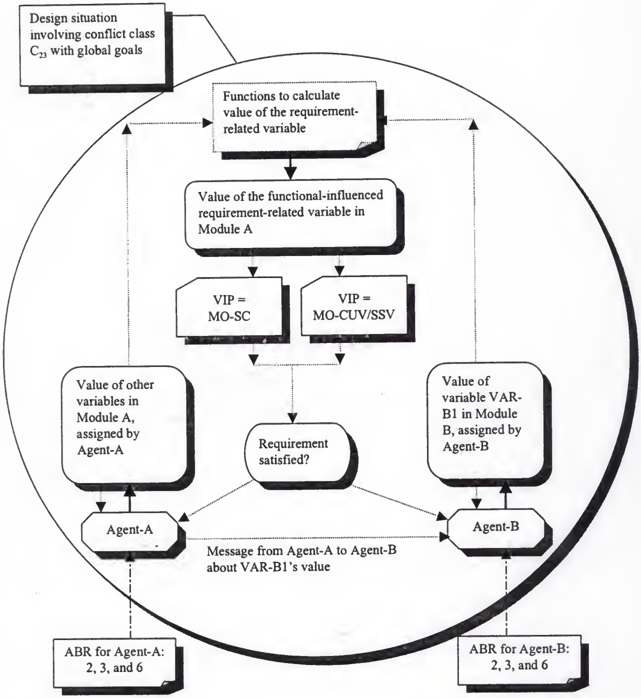
In the second case where three or more agents are involved, the values of some variables in two modules (Module B and C) are controlled by the value of a variable in the third module (i.e. VAR-A1 in Module A). Agents responsible for Module B and C (i.e. Agent-B and C) have different requests on the value of VAR-A1 due to the different perceptions. In the design, it is expected that Agent B and C send different requests to Agent-A on their desired values of VAR-A1. Upon receiving the conflicting requests from Agent-B and C, Agent-A seeks to be cooperative according to the ABM rules. Also, Agent-A tends to focus on the latest request, thinking it is more suitable to the frequently changing situation. As a result, the path shape of VAR-A1 specified by Agent-A tends to be oscillatory. Its destination could be divergent, slowly convergent, or stable at several values. The process is shown in Figure 4-26. The resulting VIPs are similar to that shown in Figure 4-10 and 4-15.

Analysis 7: Conflicting Local Goals, Global Requirement

The mapping to be analyzed (from the conflict class "conflicting local goals, global requirement" to the associated VIPs) is shown in Figure 4-11, Section 4.3.3.

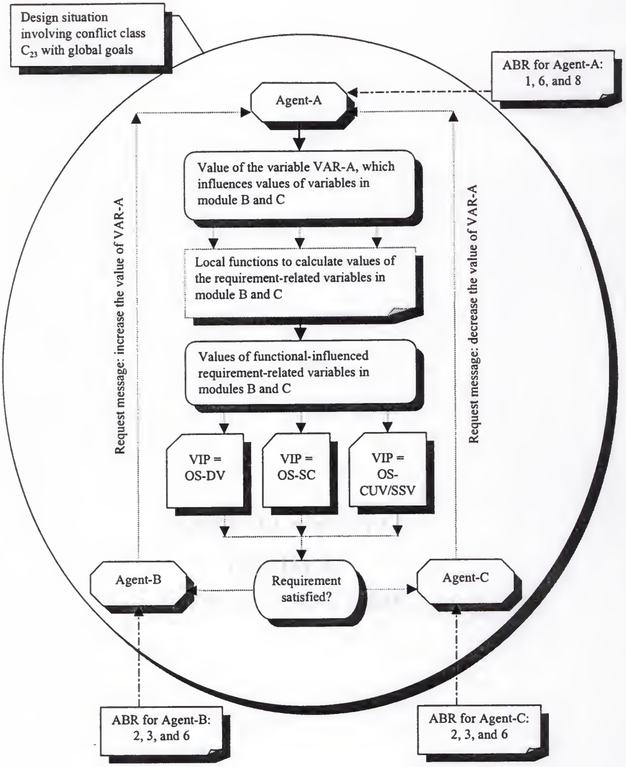
Conflicts with the fundamental cause of conflicting local goals may involve any one of the three types of requirements, i.e. aggregate, interface-compatibility, and functional-influenced. In this section, a situation involving global requirement (i.e. Class C_{31}) is studied. In the following two sections, the situations involving interface-compatibility requirement (i.e. Class C_{32}) and functional-influenced requirement (i.e. Class C_{33}) will be discussed.

To analyze the DPG of this conflict class, consider the tank design example (See also Section 3.4.3) where there is a global requirement imposed on the maximum weight of the tank. For the different agents (i.e. gunnery, structure, armor, and engine), variables related to the local goals are functionally connected to the components' weight, which contribute to the overall tank weight.


Design situation involving conflict class C_{23} with global goals
Functions to calculate value of the requirement-related variable
Value of the functional-influenced requirement-related variable in Module A
VIP = MO-SC
VIP = MO-CUV/SSV
Value of other variables in Module A, assigned by Agent-A
Requirement satisfied?
Value of variable VAR-B1 in Module B, assigned by Agent-B
Agent-A
Agent-B
Message from Agent-A to Agent-B about VAR-B1's value
ABR for Agent-A: 2, 3, and 6
ABR for Agent-B: 2, 3, and 6

(a) Two agents involved

Figure 4-26: DPG of the conflict class "different perception, functional-influenced requirement" with (a) two agents involved (b) three or more agents involved


Design situation involving conflict class C_{23} with global goals
ABR for Agent-A: 1, 6, and 8
Agent-A
Value of the variable VAR-A, which influences values of variables in module B and C
Local functions to calculate values of the requirement-related variables in module B and C
Values of functional-influenced requirement-related variables in modules B and C
VIP = OS-DV
VIP = OS-SC
VIP = OS-CUV/SSV
Request message: increase the value of VAR-A
Request message: decrease the value of VAR-A
Agent-B
Requirement satisfied?
Agent-C
ABR for Agent-B: 2, 3, and 6
ABR for Agent-C: 2, 3, and 6

(a) Three or more agents involved

Figure 4-26--Continued

In the design process, one agent (i.e. the gunnery) changes his variable of gun size to a larger value to achieve his local goal. Our model suggests that all agents will try to be cooperative, and thus that his design change will trigger a cascade of changes in the related modules' designs in the same direction (i.e. steadily increasing variable values). For instance, the vehicle structure agent wants to increase the vehicle size for more internal space to adapt the bigger gun. The armor agent has to add more armor to protect more area. The engine agent has to build a more powerful engine to provide necessary mobility for the tank and fuel capacity must be increased. These changes will in turn influence the structure agent's choice, resulting in another loop of design process. Since all these local goal-related variables are functionally related to a module variable (weight), which contributes to the global requirement-related variable (i.e. the overall tank weight), the global variable's value is predicted to steadily increase. In consequence, the global requirement variable (i.e. the overall weight of the tank) will increase beyond the required maximum value perhaps and perhaps will diverge. Figure 4-27 illustrates the DPG of the above design process, in which agents A, B, C, and D refer to the agents responsible for the gunnery, vehicle structure, armor, and engine in the tank design example. The resulting VIP is expected to be similar to that shown in Figure 4-12.

Analysis 8: Conflicting Local Goals, Interface-Compatibility Requirement

The VIPs and DPG analysis of this conflict class are similar to those of class "different perception, interface-compatibility requirement" discussed in Analysis 5.

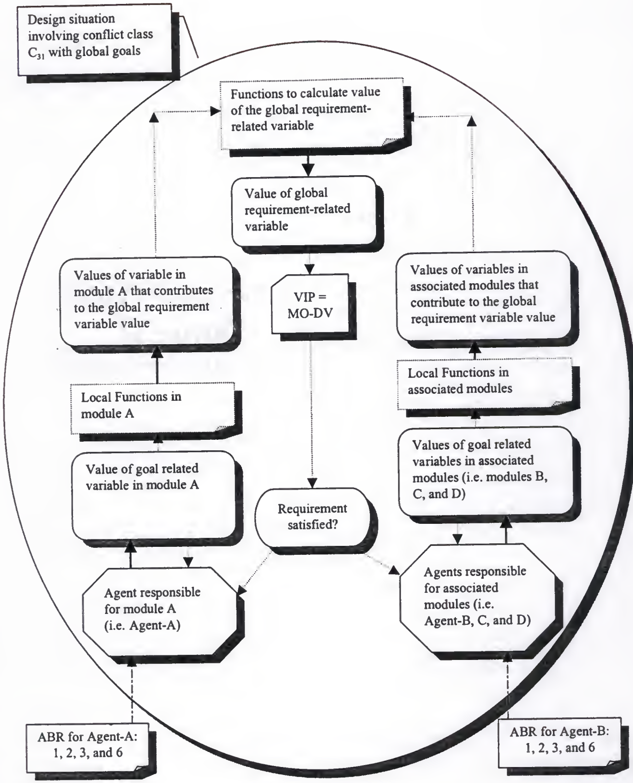

Design situation involving conflict class C31 with global goals
Functions to calculate value of the global requirement-related variable
Value of global requirement-related variable
VIP = MO-DV
Values of variable in module A that contributes to the global requirement variable value
Values of variables in associated modules that contribute to the global requirement variable value
Local Functions in module A
Local Functions in associated modules
Value of goal related variable in module A
Values of goal related variables in associated modules (i.e. modules B, C, and D)
Requirement satisfied?
Agent responsible for module A (i.e. Agent-A)
Agents responsible for associated modules (i.e. Agent-B, C, and D)
ABR for Agent-A: 1, 2, 3, and 6
ABR for Agent-B: 1, 2, 3, and 6

Figure 4-27: DPG of the conflict class "conflicting local goals, global requirement"

Here, two agents (i.e. Agent-A and B) have conflicting local goals on the interface designs of two modules (i.e. VAR-A1 in Module A and VAR-B1 in Module B). There is an interface-compatibility requirement that the two designs in the interface should be compatible. That is, the difference between variables VAR-A1 and VAR-B1 should be within a small range:

$$|VAR\text{-}A1 - VAR\text{-}B1| < \varepsilon$$

In pursuing the interface requirement, the two agents seek to be cooperative. The gap between the two designs slowly narrows monotonically. However, since the agents are primarily local goal oriented in their designs, the converging process tends to be so slow that the requirement cannot be achieved at deadline of the design process. In some cases, it may even become stuck at an unacceptable value. Therefore, the path shape of the requirement-related variable tends to be monotonic. Its destination could be slowly convergent or converging to an unacceptable value/stable at several values. The process is illustrated in Figure 4-28. The resulting VIPs are similar to those shown in Figure 4-10.

Analysis 9: Conflicting Local Goals, Functional-Influenced Requirement

The mapping from the conflict class "conflicting local goals, functional-influenced requirement (3 or more agents involved)" (Class 3.3) to the associated VIPs is shown in Figure 4-13, Section 4.3.3.

To illustrate the DPG of this class, consider the example discussed in Section 4.3.3, the agents responsible for the control and tail-structure modules (indicated by B and C respectively) have conflicting goals with respect to the location of the propulsion module (being designed by Agent-A) in that the local requirement-related variables in control and tail structure modules, i.e. controllability in yaw-direction (VAR-B1) for control module and heat/noise/vibration loading (VAR-C1) for the tail structure module, are influenced by the location of the propulsion module via functions.

In the design process, Agent-B sends request messages to Agent-A asking him to move the propulsion module close to the fuselage to achieve his (Agent-B) local goal (i.e. better yaw-controllability when one of the propulsion modules is shut down). Meanwhile, Agent-C also sends requests to Agent-A asking him to move the propulsion module away from the fuselage based on Agent-C's local goal (i.e. keeping the tail clear from the exhaust from the propulsion module for low heat/noise/vibration loading). Clearly, the request messages from Agent-B and Agent-C are conflicting.

The ABM rule predicts that Agent-A will try to be cooperative. Further, it is predicted that, upon receiving the conflicting requests from two agents, Agent-A will focus on the latest request, and tend to ignore the older one. Thus, the value of the propulsion module location (i.e. VAR-A1) selected by Agent-A tends to be oscillatory between the two values preferred by Agent-B and Agent-C. As a result, the values of the requirement-related variables in modules B and C (i.e. VAR-B1 and VAR-C1) tend to be oscillatory. The pattern of VAR-B1 and VAR-C1 could be oscillatory (in shape), divergent, slowly convergent, or stable at several values (in destination). This process is shown graphically in Figure 4-29. The resulting VIP is predicted to be as shown in Figure 4-15.

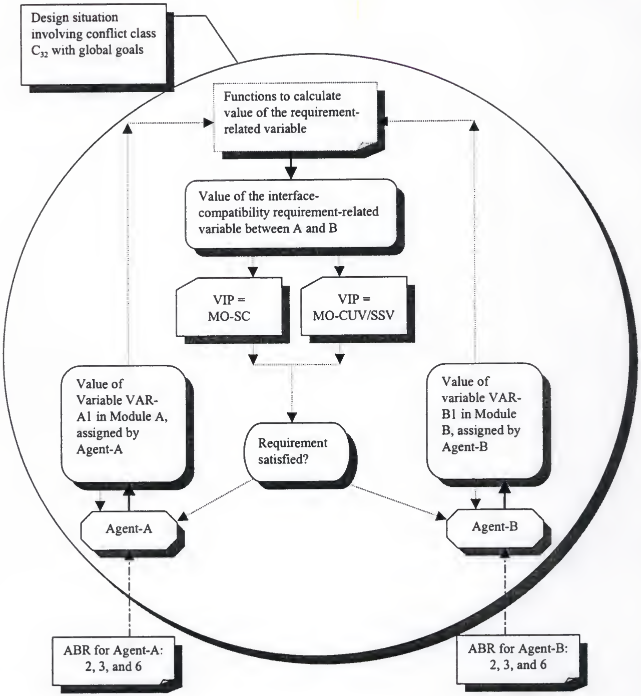

Design situation involving conflict class C_{32} with global goals
Functions to calculate value of the requirement-related variable
Value of the interface-compatibility requirement-related variable between A and B
VIP = MO-SC
VIP = MO-CUV/SSV
Value of Variable VAR-A1 in Module A, assigned by Agent-A
Requirement satisfied?
Value of variable VAR-B1 in Module B, assigned by Agent-B
Agent-A
Agent-B
ABR for Agent-A: 2, 3, and 6
ABR for Agent-B: 2, 3, and 6

Figure 4-28: DPG of the conflict class "conflicting local goals, interface-compatibility requirement"

Analysis 10: Inadequate Communication, Global Requirement

In this conflict class, several design agents (i.e. Agent-A, B, etc.) responsible for different modules (i.e. Module A, B, etc.) are involved in a global requirement.

In the design, each agent carries out his module design based on his own local goals and perception. The global requirement may not be satisfied. Realizing the global goal is not met; one agent may send messages to other agents asking them to change their designs to achieve the global requirement. However, due to the inadequate communication, the agents tend to focus on their own module designs and pay little attention of the requests from other agents or the changes of other modules' designs. According to the ABM rules, their internal designs tend to be consistent, while these designs may be based on the obsolete information and often lead to conflict with each other. Based on the ABM rules, the agents are honestly cooperative in achieving the global goals. When they are aware of the requests from other agents or the changes of other agents' designs after a long period of time, it is predicted the agents will adjust their own module-designs, often significantly, to cope with other agents' designs. Therefore, the path shape of the history values of the design results (i.e. the requirement-related variables) tends to be segmental along the design period. Its destination could be divergent, slowly convergent, or converging at an unacceptable value/stable at several values. This process is illustrated in Figure 4-30. The resulting VIPs are similar to those shown in Figure 4-17.

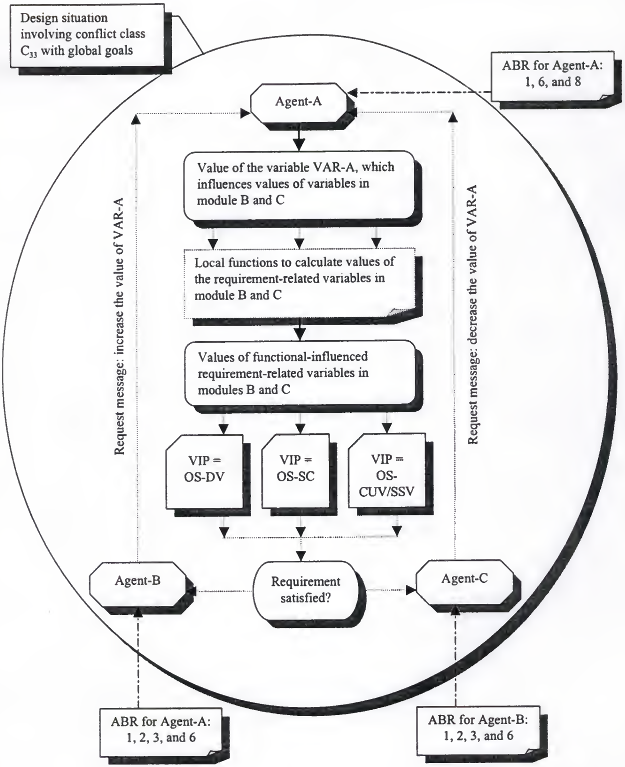

Design situation involving conflict class C_{33} with global goals
ABR for Agent-A: 1, 6, and 8
Agent-A
Value of the variable VAR-A, which influences values of variables in module B and C
Local functions to calculate values of the requirement-related variables in module B and C
Values of functional-influenced requirement-related variables in modules B and C
VIP = OS-DV
VIP = OS-SC
VIP = OS-CUV/SSV
Request message: increase the value of VAR-A
Request message: decrease the value of VAR-A
Agent-B
Requirement satisfied?
Agent-C
ABR for Agent-A: 1, 2, 3, and 6
ABR for Agent-B: 1, 2, 3, and 6

Figure 4-29: DPG of the conflict class "conflicting local goals, functional-influenced requirement (3 or more agents involved)"

Analysis 11: Inadequate Communication, Interface-Compatibility Requirement

In this conflict class, assume two agents (i.e. Agent-A and B) are responsible for designs of two modules (i.e. Module A and B). There is an interface-compatibility requirement that the variables in the interface between these two modules should match.

In the design, the interface designs in the two modules may be different due to different considerations by the two agents. The requirement is not satisfied. However, due to inadequate communication, the two agents focus on each one's module design for an excessive period of time before realizing the incompatibility between their designs in the interface. According to the ABM rules, the agents will then seek to be cooperative and adjust their designs significantly to achieve the interface-compatibility requirement. Then, the agents may focus on each one's module again for a long period of time. The inadequate communication may lead to obsolete information used in agents' module design and conflicting design result. The shape of the history values of the requirement-related variable is expected to be segmental. Its destination could be slowly convergent, or converging to an unacceptable value/stable at several values. This process is illustrated in Figure 4-31. The resulting VIPs are similar to those shown in Figure 4-17.

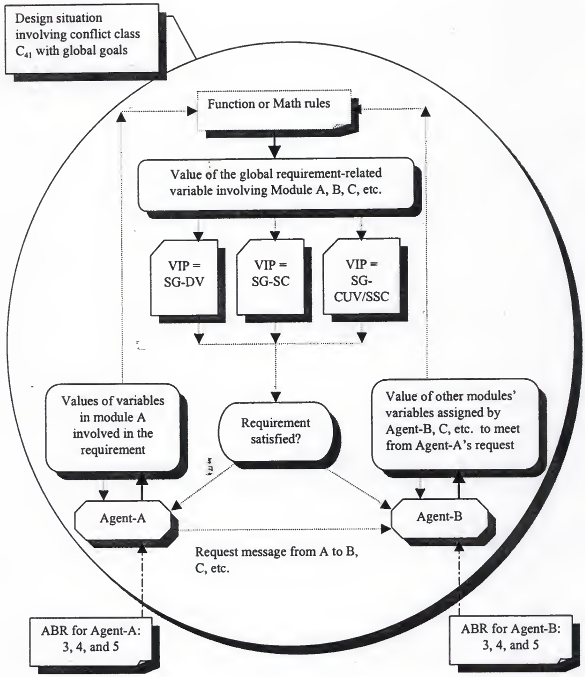

Design situation involving conflict class C_{41} with global goals
Function or Math rules
Value of the global requirement-related variable involving Module A, B, C, etc.
VIP = SG-DV
VIP = SG-SC
VIP = SG-CUV/SSC
Values of variables in module A involved in the requirement
Requirement satisfied?
Value of other modules' variables assigned by Agent-B, C, etc. to meet from Agent-A's request
Agent-A
Agent-B
Request message from A to B, C, etc.
ABR for Agent-A: 3, 4, and 5
ABR for Agent-B: 3, 4, and 5

Figure 4-30: DPG of the conflict class "inadequate communication, global requirement"

Analysis 12: Inadequate Communication, Functional-Influenced Requirement

As introduced in Section 4.3.3, the mapping from the conflict class "inadequate communication, functional-influenced requirement" to the associated VIPs is shown in Figure 4-16.

Conflicts caused by inadequate communication may involve three requirement types, i.e. the global, interface-compatibility, and functional-influenced. Here, an example involving the functional-influenced requirement (Class 4.3) is used to analyze this class.

Consider the example of aircraft design introduced in Section 4.3.3 where two agents are involved in designs of the propulsion and structure modules. The two agents are involved in a functional-influenced interaction that the exhaust of the propulsion units may cause high loading of heat, noise and vibration and severe damage on the tail-structure if improperly located. In the design process, the structure agent hopes the rear plane structure could be cleared from the propulsion units' thrust, i.e. the value of distance between the propulsion units and tail-structure in the spanwise direction should be greater than a certain value ($V > V_0$). He sends a request message to the propulsion agent about his preferred value of the propulsion units' location. Due to inadequate communication, the propulsion agent spends an excessively long period of time focusing on his own module before considering the request from the structure agent. Being cooperative, the propulsion agent then changes the propulsion units' location significantly as requested by the structure agent.

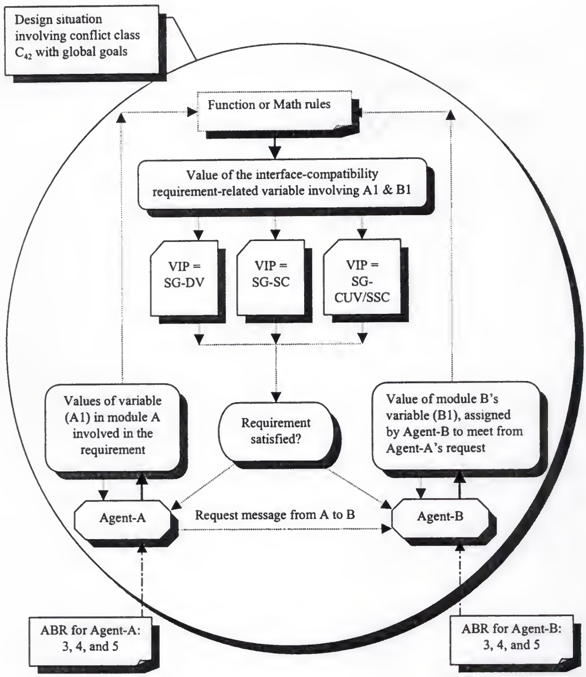
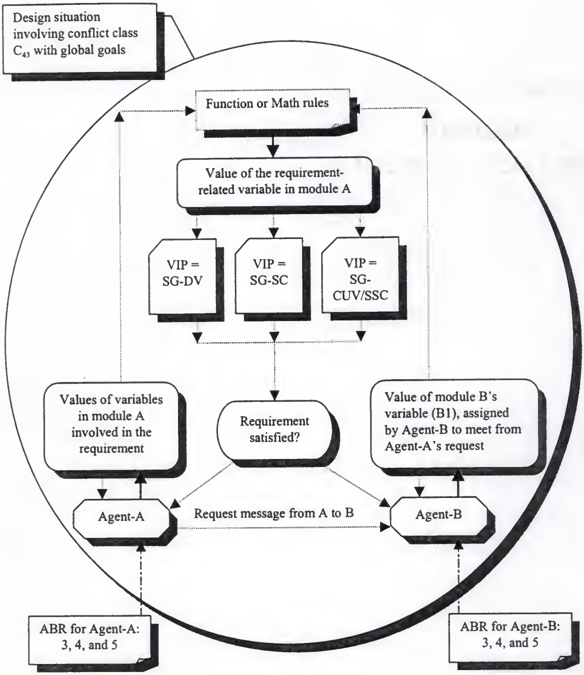

Design situation involving conflict class C_{42} with global goals
Function or Math rules
Value of the interface-compatibility requirement-related variable involving A1 & B1
VIP = SG-DV
VIP = SG-SC
VIP = SG-CUV/SSC
Values of variable (A1) in module A involved in the requirement
Requirement satisfied?
Value of module B's variable (B1), assigned by Agent-B to meet from Agent-A's request
Agent-A
Request message from A to B
Agent-B
ABR for Agent-A: 3, 4, and 5
ABR for Agent-B: 3, 4, and 5

Figure 4-31: DPG of the conflict class "inadequate communication, interface-compatibility requirement"

In the mean time, the structure agent has done some further change in his design, which the propulsion agent is unaware of for a long period of time due to the inadequate communication. Thus, the propulsion agent's design is based on the obsolete information in a long period of time before he is aware of the structure agent's changes in design and adjusts his own design. As a result, the propulsion agent's design (i.e. location value of the propulsion unit) is expected to be segmental along the time axis. Its destination could be divergent, slowly convergent, or converge to an unacceptable value/stable at several values. This process is illustrated in Figure 4-32. The resulting VIP is predicted to be as shown in Figure 4-17.

Analysis 13: Hard Problems

For hard problem, the cases involving three requirement types are similar to each other. Here, the agents are expected to be cooperative in the design according to the ABM rules. The shape of the requirement-related variable tends to be monotonic. However, the requirement-related variable does not converge towards the acceptable value because the requirement is beyond the limit of technology or resource. Rather, it becomes "stable" at an unacceptable value. Therefore, its destination tends to be converging to an unacceptable value or stable at several values.



Design situation involving conflict class C_{43} with global goals
Function or Math rules
Value of the requirement-related variable in module A
VIP = SG-DV
VIP = SG-SC
VIP = SG-CUV/SSC
Values of variables in module A involved in the requirement
Requirement satisfied?
Value of module B's variable (B1), assigned by Agent-B to meet from Agent-A's request
Agent-A
Request message from A to B
Agent-B
ABR for Agent-A: 3, 4, and 5
ABR for Agent-B: 3, 4, and 5

Figure 4-32: DPG of the conflict class "inadequate communication, functional-influenced requirement"

4.4.3 General Forward Design Process Results

As described in the above analyses, the ABM rules and derived DPGs may be used to predict in detail the evolution of variables in the design process from conflict causes to the explicit symptoms (the VIPs). The comprehensive set of results, showing the VIPs that are expected to be produced by various conflict classes, is presented in Table 4-4.

If sufficient experimental data were available, it might be possible to use statistical methods, such as data mining, to analyze various VIPs and conflict classes and determine a unique one-to-one mapping between each conflict class and its corresponding Value Indicator Patterns (Weiss 1998). Due to the limited experimental information available in this study, it has not been feasible to create a unique mapping between each conflict class and an associated VIP. However, the ABM rules and DPGs developed do describe how the conflict causes lead to explicit symptoms, and allow identification of a limited set of VIPs for each conflict class.

Table 4-4: Mapping of VIPs expected to result from various conflict classes[7]

	Global requirement (GR)	Interface-compatibility requirement (ICR)	Functional-influenced requirement (FIR)
Different terminology (DT)	CH-DV CH-SC CH-CUV/SSV	CH-DV CH-SC CH-CUV/SSV	CH-DV CH-SC CH-CUV/SSV
Different perception (DP)	MO-SC MO-CUV/SSV	MO-SC MO-CUV/SSV	MO-SC[8] MO-CUV/SSV OS-DV OS-SC OS-CUV/SSV
Conflicting local goals (CLG)	MO-DV	MO-SC MO-CUV/SSV	MO-SC[9] MO-CUV/SSV OS-DV OS-SC OS-CUV/SSV
Inadequate communication (IC)	SG-DV SG-SC SG-CUV/SSV	SG-DV SG-SC SG-CUV/SSV	SG-DV SG-SC SG-CUV/SSV
Hard problem[10] (HP)	MO-CUV/SSV	MO-CUV/SSV	MO-CUV/SSV

[7] For abbreviation of the VIPs, see Table 4-1.

[8] For possible VIPs of the conflict class DP (FIR), MO-SC and MO-CUV/SSV have resulted from the FIR that involves two agents (i.e. FIR2). OS-DV, OS-SC, and OS-CUV/SSV have resulted from the FIR that involves three or more agents (i.e. FIR3+).

[9] For possible VIPs of the conflict class CLG (FIR), MO-SC and MO-CUV/SSV have resulted from the FIR that involves two agents (i.e. FIR2). OS-DV, OS-SC, and OS-CUV/SSV have resulted from the FIR that involves three or more agents (i.e. FIR3+).

[10] The problem difficult to solve, or the goal difficult to achieve due to technology limit. It is assumed that the design agents keep on trying to cooperate, adjust their variable values, and resolve the conflict before reaching a hard problem, where the requirement-related variable value is monotonic and converges to an unacceptable value.

4.5 Inverse Process of Conflict Cause Identification

4.5.1 Overview of the Conflict Cause Identification Process

In the previous section, the Design Process Graph (DPG), which describes how variables, relations and requirements evolve in forward design processes, is presented. Also, the mappings from conflict classes (i.e. combinations of conflict fundamental causes and requirement types) to associated Value Indicator Patterns (VIPs) are identified and studied. This section will focus on the Inverse Identification Process (I^2P) to identify the conflict causes from the VIP observed.

Based on literature searched (Hayes-Roth 1983, Carrico 1989) and a personal communication with Dr. D. Dankel[11], it is concluded that satisfactory methods for inverting rule-based expert systems have yet to be developed. It is feasible, however, to invert Table 4-4 to obtain Table 4-5. Therefore in this work, the C^2I agent will use the mappings from VIPs to conflict classes shown in Table 4-5 to identify the conflict classes and causes.

As Figure 4-2 in Section 4.3.1 illustrates, to begin the I^2P, the C^2I agent monitors the real-time behavior of requirement-related public variable values. When a conflict is detected, the C^2I agent then retrieves the design history (i.e. past and present values of variables) from the Public Design Information (PDI) database where it has been stored, identifies the VIPs, obtains the requirement type(s) from the PDI database, and identifies the conflict classes using the mappings of Table 4-5.

[11] Dr. Douglas D. Dankel is an Assistant Professor in Computer and Information Science and Engineering department, University of Florida.

Table 4-5: Mapping of conflict classes which could produce various VIPs

	Divergent	Slow convergent	Converge to an unacceptable value/stable at several values	Converge to an acceptable value
Monotonic	CLG (GR)	DP (GR) DP (ICR) DP (FIR2)[12] CLG (ICR) CLG (FIR2)	HP DP (ICR) DP (FIR2) CLG (ICR) CLG (FIR2)	No conflict. No HP.
Oscillatory	CLG (FIR 3+)[13] DP (FIR3+)	CLG (FIR3+) DP (FIR3+)	CLG (FIR3+) DP (FIR3+)	No conflict. No HP.
Chaotic	DT (GR) DT (ICR) DT (FIR)	DT (GR) DT (ICR) DT (FIR)	DT (GR) DT (ICR) DT (FIR)	No conflict. No HP.
Segmental	IC (GR) IC (ICR) IC (FIR)	IC (GR) IC (ICR) IC (FIR)	IC (GR) IC (ICR) IC (FIR)	No conflict. No HP.

According to the ABM rules, it is expected that the design agents keep on trying to be cooperative, adjust their designs to resolve the conflict before ending up with a hard problem, i.e. a goal difficult to achieve because of technology limit. Therefore, the destination of the requirement-related variable value is expected to be converging to an unacceptable value rather than divergent or slowly convergent.

In this work, we cannot tell Hard Problem from other conflict causes from the VIP of MO-CUV/SSV.

[12] The Functional-Influenced Requirement that involves two parties, i.e. the design agents.

[13] The Functional-Influenced Requirement that involves three or more parties, i.e. the design agents.

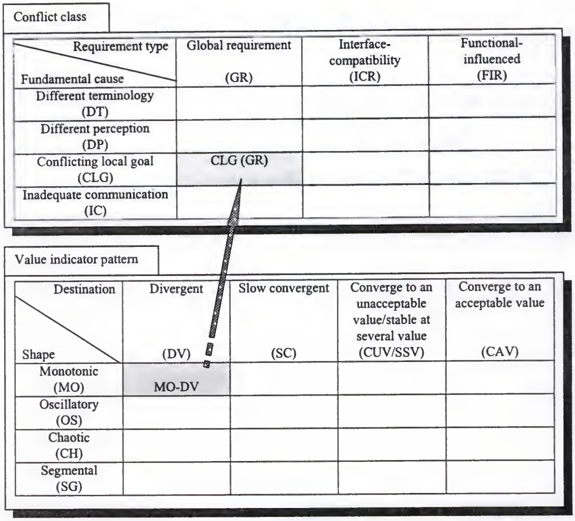
4.5.2 Use of the Inverse Identification Process

In the Inverse Identification Process (I^2P), when a conflict is detected, the C^2I agent then analyze the public data (i.e. past and present values of the public requirement-related variables) to determine the VIPs. Then, the C^2I agent uses the mappings from VIP/requirement type to the associated conflict causes (Table 4-5) to identify a set of possible classes.

In the current C^2I system, the VIP/requirement type-conflict cause mappings listed in Table 4-5 are not always unique. In some cases, the mapping is unique one-to-one mapping. In other cases, a one-to-several mapping is obtained. Here, two examples are used to illustrate the two cases.

VIP Example 1: "Monotonic, Divergent" VIP/Global Requirement

In this example, when a conflict is detected, the C^2I agent retrieves the history values of the public requirement-related variables and identifies the VIP as "monotonic, divergent" (MO-DV). Also, the requirement type is retrieved from the Public Design Information (PDI) database and determined to be global. Referring to the mapping from VIP/requirement type to associated conflict causes illustrated in Table 4-5, the C^2I agent finds one matching conflict cause as "conflicting local goals" (CLG). This is a one-to-one mapping. Thus, the conflict class is identified uniquely as "conflicting local goals, global requirement". The one-one mapping from "monotonic, divergent/global requirement" to the associated conflict cause (i.e. conflicting local goals) is graphically illustrated in Figure 4-33.



Conflict class

Requirement type / Fundamental cause	Global requirement (GR)	Interface-compatibility (ICR)	Functional-influenced (FIR)
Different terminology (DT)			
Different perception (DP)			
Conflicting local goal (CLG)	CLG (GR)		
Inadequate communication (IC)			

Value indicator pattern

Destination / Shape	Divergent (DV)	Slow convergent (SC)	Converge to an unacceptable value/stable at several value (CUV/SSV)	Converge to an acceptable value (CAV)
Monotonic (MO)	MO-DV			
Oscillatory (OS)				
Chaotic (CH)				
Segmental (SG)				

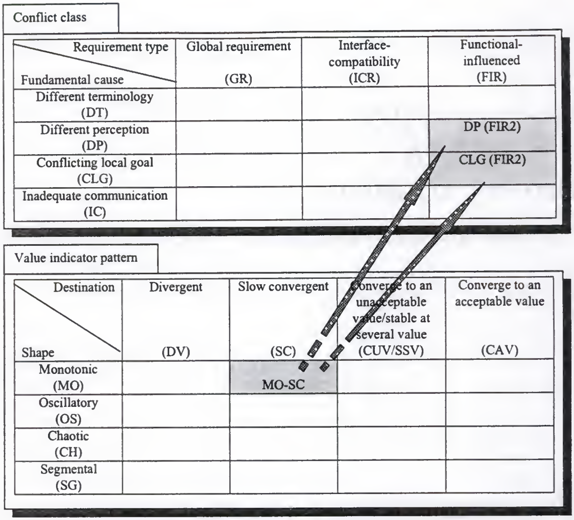
Figure 4-33: Mapping from "monotonic, divergent/global requirement" to the associated conflict cause(s) (i.e. conflicting local goals)

<u>VIP Example 2: "Monotonic, Slowly Convergent" VIP/Functional-Influenced Requirement</u>

In this example, when a conflict is detected, the C^2I agent calculates the history values of the public requirement-related variables and identifies the VIP as "monotonic, slowly convergent" (OS-SC). Also, the requirement type is obtained as functional-influenced involving two agents (FIR2). Referring to the mapping from VIP/requirement type to associated conflict causes illustrated in Table 4-5, the C^2I agent finds two matching conflict causes as "conflicting local goals" (CLG) and "different perception". This is a one-to-several mapping. Thus, the conflict class is identified as "conflicting local goals, functional-influenced requirement" or "different perception, functional-influenced requirement". The one-to-several mapping from "monotonic, slowly convergent / functional-influenced requirement involving two agents" to the associated conflict causes (i.e. conflicting local goals or different perception) is graphically illustrated in Figure 4-34.

4.5.3 Summary

Utilizing the mapping of Table 4-5 from the VIPs and requirement types to the associated conflict classes provides a feasible approach to Conflict Cause Identification in CE design.



Conflict class

Requirement type / Fundamental cause	Global requirement (GR)	Interface-compatibility (ICR)	Functional-influenced (FIR)
Different terminology (DT)			
Different perception (DP)			DP (FIR2)
Conflicting local goal (CLG)			CLG (FIR2)
Inadequate communication (IC)			

Value indicator pattern

Destination / Shape	Divergent (DV)	Slow convergent (SC)	Converge to an unacceptable value/stable at several value (CUV/SSV)	Converge to an acceptable value (CAV)
Monotonic (MO)		MO-SC		
Oscillatory (OS)				
Chaotic (CH)				
Segmental (SG)				

Figure 4-34: Mapping from "monotonic, slowly convergent / functional-influenced requirement (two agents involved)" to the associated conflict causes (i.e. conflicting local goals, or different perception)

CHAPTER 5
SYSTEM IMPLEMENTATION

A Conflict Cause Identification (C^2I) software agent has been developed to monitor the design situation and identify the conflict class and cause when a conflict is detected. This chapter describes this C^2I software agent and the Plane World system, which is developed to provide a web-based design environment to demonstrate and validate the C^2I method developed in this work.

5.1 The C^2I Software Agent

This section focuses on the Conflict Cause Identification (C^2I) software agent, which is designed to autonomously monitor the design process, identify the conflict causes (when a conflict is detected by the CD agent), and notify human design agents, thus, supporting them in conflict resolution.

The C^2I agent has the following functions:

- Monitor the ongoing design process constantly via the public database: every time an item (i.e. a variable value) is modified, the C^2I agent will catch it and store it in the Public Design Information (PDI) database.
- When a conflict is detected (by the CD agent), calculate and identify the Value Indicator Patterns (VIPs), i.e. the path shape and destination of the requirement-related variables.

- Determine whether it is a hard problem or a conflict. For a conflict, identify its class and cause from the VIPs observed and the requirement-type available.
- Notify design agents of the conflict class and cause.

These functions are described in following sub-sections. The C^2I procedure is illustrated in Figure 5-1.

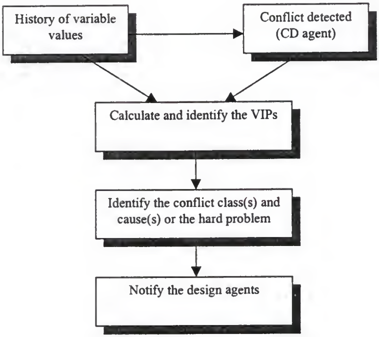

History of variable values
Conflict detected (CD agent)
Calculate and identify the VIPs
Identify the conflict class(s) and cause(s) or the hard problem
Notify the design agents

Figure 5-1: The C^2I processes

5.1.1 Monitor the On-going Design Process

To monitor the on-going design process, a specific table in the PDI database (i.e. RVHV table) is created to store the Requirement-related Variable History Values (RVHV). As Figure 5-2 illustrates, each item in the RVHV contains four parts: object, variable name, value, and timestamp. In the design process, every time a requirement-related variable's value is published, a respective item is inserted into the RVHV table. In retrieving a requirement-related variable's history, the respective items can be sorted in ascending or descending order.

Object	Variable Name	Value	Timestamp

Figure 5-2: An item in the RVHV table in database

The information of requirements (i.e. requirement types) is also stored in the PDI database for conflict cause identification.

5.1.2 Calculating the VIP

As presented in Section 4.3.2, the VIPs are partitioned along two basic dimensions: shape and destination. Here we focus on the algorithm to calculate the shape and destination of a requirement-related variable (simply called the variable in this section).

Shape

In this study, four principal types of path shapes of a variable are suggested: monotonic, oscillatory, segmental, and chaotic. A decision-tree approach (Raiffa 1970) is applied in determine whether a variable's path shape is one of these four type, as Figure 5-3 illustrates.

This section will present the algorithm for each of the four path shapes.

Identify the Segmental shape. In this work, a variable's history values are required to have the following characteristic if it is of the segmental shape:

- There is at least one singular point[1], i.e. Number $_{\text{Singular-Point(s)}} > 0$,
- The pulse points, value points similar to the singular point but jump back immediately after it jump away, are not considered as singular points, as Figure 5-4 illustrates,
- Within the segments, the value difference between any adjacent points should be much less than that of the singular points.

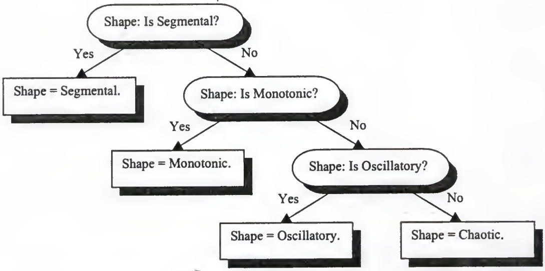
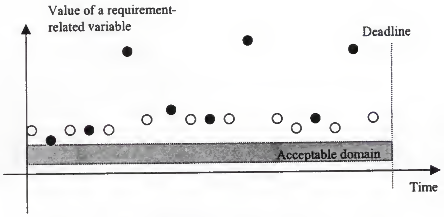


Figure 5-3: Decision tree for determining the path shape

To determine if the variable's values are of the segmental shape, the C^2I agent first calculates the feature values (i.e. Number $_{\text{Singular-Point(s)}}$). Then, the C^2I agent verifies if all of the above characteristic conditions are met using the decision tree as illustrated in Figure 5-5.

[1] A singular point (V_i) is a value point in the design history whose value difference from its predecessor (V_{i-1}) is much greater than the value difference between its predecessor and the predecessor's predecessor (V_{i-2}), i.e. $| V_i - V_{i-1} | >> | V_{i-1} - V_{i-2} |$.



○	Value of the requirement-related variable resulted from Agent-A's design changes
●	Value of the requirement-related variable resulted from Agent-B's design changes

Figure 5-4: An example with pulses in history values of the requirement-related variable

Identify the Monotonic shape. In this study, a variable's history values are required to have the following characteristics if they are of the monotonic shape:

- There is no singular point,
- If the history values are not the same, the majority of value points in history should be either ascending or descending, i.e. Number $_{\text{Ascending}}$ > a strong majority number of history value points[2] or Number $_{\text{Descending}}$ > a strong majority number of history value points.

[2] In this work, the strong majority is considered as 75% of the variable's history value points.

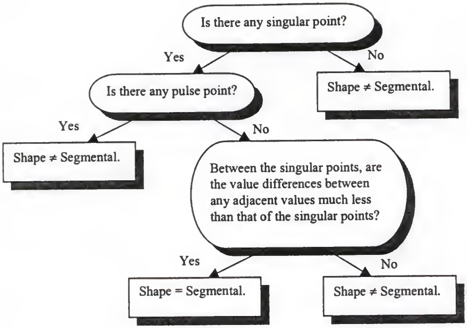

Is there any singular point?
Yes
No
Shape ≠ Segmental.
Is there any pulse point?
Yes
No
Shape ≠ Segmental.
Between the singular points, are the value differences between any adjacent values much less than that of the singular points?
Yes
No
Shape = Segmental.
Shape ≠ Segmental.

Figure 5-5: Decision tree for determining the segmental shape

To identify the monotonic shape, the C^2I agent first calculates the feature values (i.e. number of singular points). Then, it verifies if all the above characteristic conditions are met. If so, the shape is monotonic. Otherwise, the shape is non-monotonic.

The process is illustrated using a decision tree in Figure 5-6.

Identify the Oscillatory shape. In this study, the history values of a variable are required to have the following characteristics if it is of the oscillatory shape:

- There are at least one turning point(s)[3], i.e. $\text{Number}_{\text{Turning-Point(s)}} > 0$,
- There is no singular point, i.e. $\text{Number}_{\text{Singular-Point(s)}} = 0$,

[3] A turning point (V_i) is a value point in design history that the difference between the point and its predecessor and the difference between the point and its decedent are of different sign, i.e. $(V_i - V_{i-1}) * (V_{i+1} - V_i) < 0$.

- Neither the number of ascending points (i.e. Number $_{Ascending}$) nor the number of descending points (Number $_{Descending}$) is a strong majority of history value points,
- The period of the oscillatory path remains near constant or varies within an acceptable value, as Figure 5-7 illustrates.

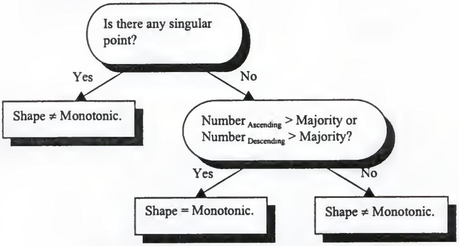

Is there any singular point?
Yes
No
Shape ≠ Monotonic.
Number Ascending > Majority or Number Descending > Majority?
Yes
No
Shape = Monotonic.
Shape ≠ Monotonic.

Figure 5-6: Decision tree for determining the monotonic shape

Based on the variable's history values, the C^2I agent calculates the associated features (i.e. number of turning points and singular points) to verify if the above requirements are satisfied. If all of them are satisfied, the variable's history values are of oscillatory shape. Otherwise, the variable's history values are not of oscillatory shape.

The process is presented using a decision tree illustrated in Figure 5-8.

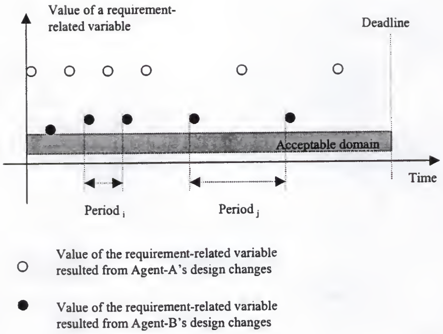

Value of a requirement-related variable
Deadline
Acceptable domain
Time
Period i
Period j
Value of the requirement-related variable resulted from Agent-A's design changes
Value of the requirement-related variable resulted from Agent-B's design changes

Figure 5-7: An example that the period of oscillatory path varies significantly

<u>Identify the Chaotic shape</u>. As illustrated in Figure 5-3, the shape of a variable's history values is of chaotic if it is not one of the other three types.

<u>Destination</u>

In this work, four types of destinations are suggested: divergent, slowly convergent, converge to an unacceptable value/stable at several values, and convergent.

To determine the destination of a requirement-related variable in the design process, the C^2I agent first calculates its predicted value at the deadline (i.e. Predicted-Value) using the least square approach adapted from Zhuang's work (Zhuang 1999) as Figure 5-9 illustrates.

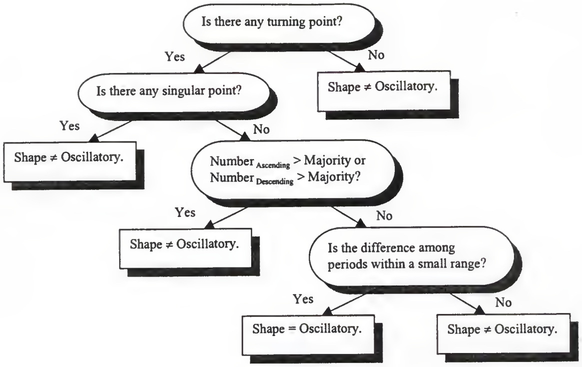

Is there any turning point?
Yes
No
Shape ≠ Oscillatory.
Is there any singular point?
Yes
No
Shape ≠ Oscillatory.
Number $_{Ascending}$ > Majority or Number $_{Descending}$ > Majority?
Yes
No
Shape ≠ Oscillatory.
Is the difference among periods within a small range?
Yes
No
Shape = Oscillatory.
Shape ≠ Oscillatory.

Figure 5-8: Decision tree for determining the oscillatory shape

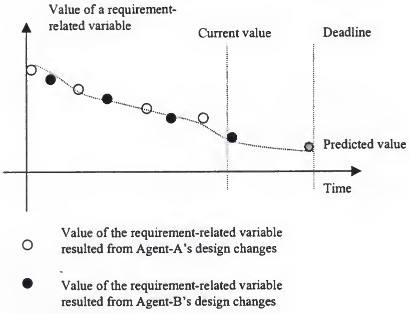


Figure 5-9: Predicted value

If the predicted value satisfies the requirement, the variable's destination is considered as Convergent.

In cases where the predicted value does not meet the requirement, the C^2I agent then calculates the absolute difference between the predicted value and the acceptable value (Equation 5-1) and compares it with the absolute difference between current value and the acceptable value (Equation 5-2). If the former value is very close to the later one (Equation 5-3), the variable's destination is identified as converge to an unacceptable value/stable at several values. Otherwise, if the former value is smaller than the later one, the variable's destination is identified as slowly convergent. If none of the above conditions is met, the destination is identified as divergent.

$$Difference_{Predicted-Acceptable} = |Value_{Acceptable} - Value_{Predicted}| \tag{5-1}$$

$$Difference_{Current-Acceptable} = |Value_{Acceptable} - Value_{Current}| \tag{5-2}$$

$$|Difference_{Predicted-Acceptable} - Difference_{Current-Acceptable}| < \varepsilon \tag{5-3}$$

The above process can be presented using a decision tree in Figure 5-10.

5.1.3 Identify the Conflict Cause

When the VIP is calculated, the C^2I agent then obtains the requirement type from the PDI database, looks up the table of mappings from VIP/requirement types to associated conflict causes (Table 4-5), and identifies the possible conflict causes.

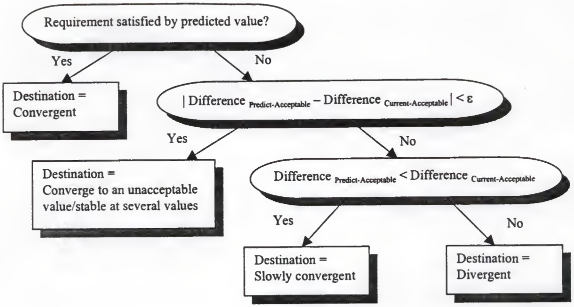

Requirement satisfied by predicted value?
Yes
No
Destination = Convergent
| Difference Predict-Acceptable – Difference Current-Acceptable | < ε
Yes
No
Destination = Converge to an unacceptable value/stable at several values
Difference Predict-Acceptable < Difference Current-Acceptable
Yes
No
Destination = Slowly convergent
Destination = Divergent

Figure 5-10: Decision tree for determining the destination of a requirement-related variable

5.1.4 Notify the Design Agents

In notifying design agents about the conflict class and cause, the C^2I agent currently sends information to all design agents involved in the design. It is recognized that in a complex real-world design system, it would likely be desirable to send the information regarding a specific conflict class and cause only to those agents that are directly involved in the conflict. This improvement would be an appropriate future development.

In the next section, the Plane World system, a web-based design system used to demonstrate and test the C^2I agent is described.

5.2 Plane World System

5.2.1 Description of the 2-D Plane Model in the Plane World System

In the Plane World simulation, multiple human agents cooperate in the preliminary conceptual design of a simplified 2-D cargo plane in a distributed CE environment. In the design, the human design agents interact via the web and could be at different graphical locations. Here the simulation is used to demonstrate and validate the C^2I methods discussed in previous chapters.

Structure of the 2-D Plane

In the real world, a plane is composed of many types of modules (Anderson 1989). In this study, modules in a 2-D cargo plane (simply called the plane) are limited to six types, i.e. Cargo, FuelTank, Propulsion, Structure, Aerodynamic, and Control[4] (Figure 5-11).

To keep the model simple, a number of important considerations, such as avionics and navigation, are omitted. The simulation only involves module-level design in this study.

Attributes and Functions of the Plane and Modules

In this study, object-oriented methods (Gamma 1995) are applied to describe the design objects, i.e. the plane, modules, and components. Each design object has its own attributes and functions.

[4] In this work, the control module is highly simplified, and is only concerned with yaw-controllability.

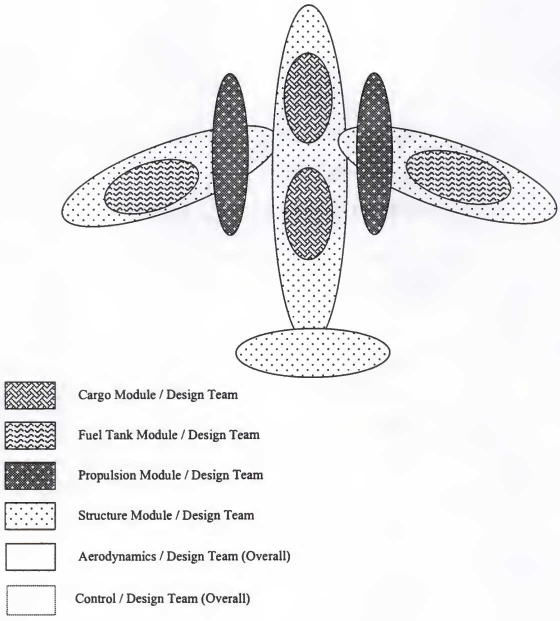

Cargo Module / Design Team
Fuel Tank Module / Design Team
Propulsion Module / Design Team
Structure Module / Design Team
Aerodynamics / Design Team (Overall)
Control / Design Team (Overall)

Figure 5-11: Modules in the 2-D cargo plane

As Table 5-1 shows, attributes of a design object are used to describe its state information (Gamma 1995). Here, functions describe the internal relations among attributes within a design object, as Table 5-2 shows.

Table 5-1: Attributes of plane and modules in the 2-D plane[5]

OBJECT	ATTRIBUTES
Plane	Speed, Range, CargoWeight, Weight, Cost, Area, DLWPBNHSpacing, DRWPBNHSpacing, DXLeftProp, DXRightProp, DXFuselageLProp, DXFuselageRProp
Cargo Module	X, Y, Width, Height, Area, CargoWeight, CargoWeightCoeff, Weight, WeightCoeff, Cost, CostCoeff
Fuel Tank Module	X, Y, Width, Height, Area, Fuel, FuelCoeff, Weight, WeightCoeff, Cost, CostCoeff
Propulsion Module	X, Y, Thrust, TSFC, FuelRate, Weight, ThrustWeightRatio, Cost, CostThrustCoeff, BNHoleSpacing
Structure Module	X, Y, Width, Height, Area, Weight, WeightCoeff, Cost, CostCoeff, Strength, BNHoleSpacing, DXLeftProp, DXRightProp
Aerodynamic	EffectiveArea, Lift, LiftCoeff, Drag, DragCoeff
Control	DXFuselageLProp, DXFuselageRProp

Relationships

In Plane World, relationships are used to describe the interactions between design objects, i.e. plane and modules. In real-world aircraft design, the relationships could be very complex. Here relationships are highly simplified as Table 5-3 shows[6].

[5] See Glossary for definitions and descriptions of the attributes.

[6] Children.VAR represents summation of the value of the Variable VAR of all the children. Meanwhile, the subscript represents the specific module. For example, $\text{Structure}_{\text{Tail}}$ represents the tail-structure module.

Table 5-2: Module functions in the 2-D plane

MODULES	FUNCTIONS
Aerodynamic	$Aerodynamic.Lift = Aerodynamic.LiftCoeff \times Aerodynamic.Area$
	$Aerodynamic.Drag = Aerodynamic.DragCoeff \times Aerodynamic.Area$
Cargo	$Cargo.Area = Cargo.Width \times Cargo.Height$
	$Cargo.CargoWeight = Cargo.CargoCoeff \times Cargo.Area$
	$Cargo.Weight = Cargo.WeightCoeff \times Cargo.Area$
	$Cargo.Cost = Cargo.CostCoeff \times Cargo.Area$
	$Cargo.CostCoeff = Cargo.CostCoeff_{default} \times \left(\frac{Cargo.WeightCoeff_{present}}{Cargo.WeightCoeff_{default}} \right)^{-1}$
FuelTank	$FuelTank.Area = FuelTank.Width \times FuelTank.Height$
	$FuelTank.Fuel = FuelTank.FuelCoeff \times FuelTank.Area$
	$FuelTank.Weight = FuelTank.WeightCoeff \times FuelTank.Area$
	$FuelTank.Cost = FuelTank.CostCoeff \times FuelTank.Area$
	$FuelTank.CostCoeff = FuelTank.CostCoeff_{default} \times \left(\frac{FuelTank.WeightCoeff_{present}}{FuelTank.WeightCoeff_{default}} \right)^{-1}$

Propulsion	$Propulsion.FuelRate = Propulsion.Thrust \times Propulsion.TSFC$
	$Propulsion.Weight = \frac{Propulsion.Thrust}{Propulsion.ThrustWeightRatio}$
	$Propulsion.Cost = Propulsion.Thrust \times Propulsion.CostCoeff$
	$Propulsion.CostCoeff = Propulsion.CostCoeff_{default} \times \left(\frac{Propulsion.TSFC_{present}}{Propulsion.TSFC_{default}}\right)^{-1} \times \left(\frac{Propulsion.ThrustWeightRatio_{present}}{Propulsion.ThrustWeightRatio_{default}}\right)$
Structure	$Structure.Area = Structure.Width \times Structure.Height$
	$Structure.Weight = Structure.WeightCoeff \times Structure.Area$
	$Structure.Cost = Structure.CostCoeff \times Structure.Area$
	$Structure.CostCoeff = Structure.CostCoeff_{default} \times \left(\frac{Structure.WeightCoeff_{present}}{Structure.WeightCoeff_{default}}\right)^{-1} \times \left(\frac{Structure.Strength_{present}}{Structure.Strength_{default}}\right)$

Requirements

A requirement is a constraint that must be satisfied at the completion of a design. As Table 5-4 shows, three principal types of requirements are allowed in Plane World, i.e. global, interface-related, and functional-influenced requirements. Numerical values are used in Table 5-4 thus providing an example of a requirement set.

Table 5-3: Relationships in the 2-D plane

OBJECT	RELATIONSHIPS
Plane	$Plane.Speed = \left(\frac{Propulsion.Thrust}{Aerodynamic.Drag}\right)^{1/2}$
	$Plane.Range = Plane.Speed \times \left(\frac{FuelTank.Fuel}{Propulsion.FuelRate}\right)$
	$Plane.CargoWeight = Cargo.CargoWeight$
	$Plane.Weight = Children.Weight$
	$Plane.Cost = Children.Cost$
	$Plane.Area = Structure.Area$
	$Plane.DXTailLeftProp = \left(Structure_{Tail}.X - \frac{1}{2} Structure_{Tial}.Width\right) - \left(Propulsion_{Left}.X + \frac{1}{2} Propulsion_{Left}.Width\right)$
	$Plane.DXTailRightProp = \left(Propulsion_{Right}.X - \frac{1}{2} Propulsion_{Right}.Width\right) - \left(Structure_{Tail}.X + \frac{1}{2} Structure_{Tial}.Width\right)$
	$Plane.DXFuselageLProp = Structure_{Fuselage}.X - Propulsion_{Left}.X$
	$Plane.DXFuselageRProp = Propulsion_{Right}.X - Structure_{Fuselage}.X$
	$Plane.DLWPBNHSpacing =$ $Propulsion_{Left}.BNHSpacing - Structure_{LeftWing}.BNHSpacing$
	$Plane.DRWPBNHSpacing =$ $Propulsion_{Right}.BNHSpacing - Structure_{RightWing}.BNHSpacing$

Cargo	*Cargo.CargoWeight = Children.CargoWeight*
	Cargo.Weight = Children.Weight
	Cargo.Cost = Children.Cost
FuelTank	*FuelTank.Fuel = Children.Fuel*
	FuelTank.Weight = Children.Weight
	FuelTank.Cost = Children.Cost
Propulsion	*Propulsion.Thrust = Children.Thrust*
	Propulsion.FuelRate = Children.FuelRate
	Propulsion.Weight = Children.Weight
	Propulsion.Cost = Children.Cost
Structure	*Structure.Weight = Children.Weight*
	Structure.Cost = Children.Cost
	Structure.Area = Children.Area
	$Structure_{Tail}$*.DXLeftProp = Plane.DXLeftProp*
	$Structure_{Tail}$*.DXRightProp = Plane.DXRightProp*
Aerodynamic	*Areodynamic.Area = Plane.Area*
Control	*Control.DXFuselageLProp = Plane.DXFuselageLProp*
	Control.DXFuselageRProp = Plane.DXFuselageRProp

Table 5-4: Requirements in Plane World

REQUIREMENT TYPE	REQUIREMENT
Global requirement	$Plane.Speed > 500\ mph$
	$Plane.Range > 2{,}000\ miles$
	$Plane.CargoWeight > 10{,}000\ lb.$
	$Plane.Weight < 25{,}000\ lb.$
	$Plane.Cost < \$10\ million$
Interface-related requirement	$Plane.DLWPBNHSpacing < 0.125\ in.$
	$Plane.DRWPBNHSpacing < 0.125\ in.$
Functional-influenced requirement	$Structure_{Tail}.DXLeftProp > 0\ ft.$
	$Structure_{Tail}.DXRightProp > 0\ ft.$
	$Control.DXFuselageLProp < 38\ ft.$
	$Control.DXFuselageRProp < 38\ ft.$

5.2.2 Architecture of the Plane World Design and C^{2}I System

This section focuses on two issues, i.e. the web-based design and Conflict Cause Identification (C^{2}I) processes in the Plane World system and the distributed structure of the Plane World system.

Web-based Design and C^2I Process

As Figure 5-12 illustrates, in the web-based Plane World simulation, six design agents, a software C^2I agent, and a software conflict-detection (CD) agent[7] cooperate in a design task to develop an abstract 2-D cargo plane (at module level), which satisfies requirements listed in Table 5-4. Each design agent has a design window (similar to the one shown in Figure 5-13) to specify the variable values for his modules and to view the design results for his own module and for the other agents' modules and the plane overall. In the example design tasks, independent humans (i.e. the author and several other graduate students) are the agents responsible for designing various different modules of the 2-D plane. Note that while these design agents were all on campus, though on different terminals, communication was via the web and they could as well have been located far apart.

The C^2I software agent constantly monitors the design process, identifies the conflict classes and causes once a conflict is detected by the CD agent, and notifies the design agents, thus, supporting the design agents toward conflict resolution.

Distributed Structure of the Web-based Plane World System

This section focuses on the distributed structure of the Plane World system.

As Figure 5-14 illustrates, the Plane World system utilizes an object-oriented client-server structure (Tanenbaum 1994, Grimes 1997) to coordinate the computation and communication in a distributed environment.

[7] This work focuses on the C^2I process, and uses Zhuang's CD agent (Zhuang 1999).

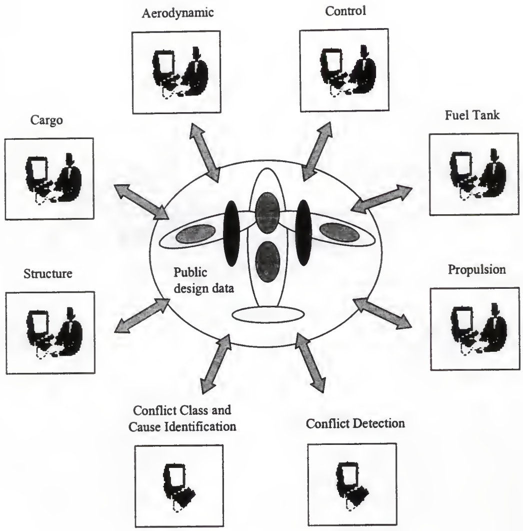
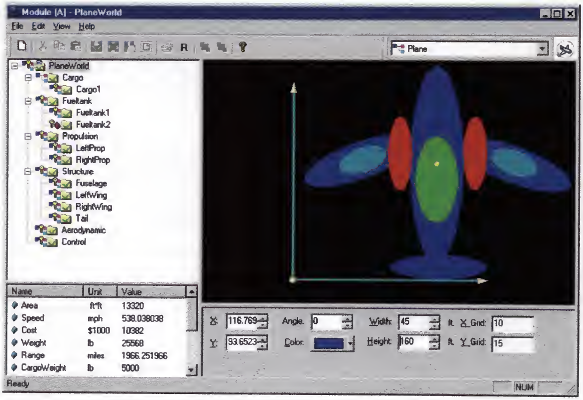


Figure 5-12: Human design agents and software agents involved in the Plane World system



- PlaneWorld
  - Cargo
    - Cargo1
  - Fueltank
    - Fueltank1
    - Fueltank2
  - Propulsion
    - LeftProp
    - RightProp
  - Structure
    - Fuselage
    - LeftWing
    - RightWing
    - Tail
  - Aerodynamic
  - Control

Name	Unit	Value
Area	ft*ft	13320
Speed	mph	538.038038
Cost	$1000	10382
Weight	lb	25568
Range	miles	1966.251966
CargoWeight	lb	5000

X: 116.769 Angle: 0 Width: 45 ft X Grid: 10
Y: 93.6523 Color: Height: 160 ft Y Grid: 15

🟩	Cargo
🟦	Fueltank
🟥	Propulsion
🟦	Structure

Figure 5-13: A design window in Plane World system[8]

[8] Note that while the design agent could view public variable values of other modules, he can only specify the variable values for his own module. Also, the design requirements, functions, and relationships are loaded into the system by the administrator before the design.

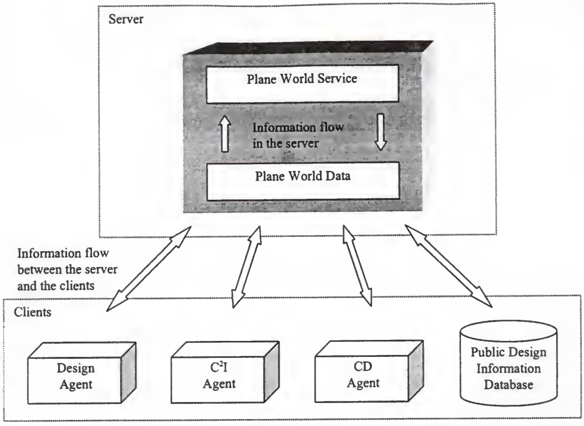

Server
Plane World Service
Information flow in the server
Plane World Data
Information flow between the server and the clients
Clients
Design Agent
C²I Agent
CD Agent
Public Design Information Database

Figure 5-14: Distributed structure of the Plane World system

At the server level, a global server is used to carry out the communication among agents and the database operations. It involves two objects, i.e. Plane World Service and Plane World Data.

- Plane World Service: implements the detailed, low-level functions of communication and database operations.
- Plane World Data: serves as a mediator between the Plane World Service and the clients. In Plane World Data, various different interfaces are created for different clients to communicate with the server.

At the client level, the Plane World system involves the following client objects:

- Design agents, independent agents (currently human) responsible for the designs of different modules of a 2-D plane seeking to satisfy all the design requirements listed in Table 5-4.
- Conflict Cause Identification (C^2I) agent: A software agent that extracts the Value Indicator Patterns (VIPs) from design information (i.e. present and past values of variables), gets the requirement types, and identifies the conflict classes and causes when the conflict is detected.
- Conflict detection (CD) agent: A software agent that monitors the ongoing design processes, predicts and detects the existence of conflicts. This agent is adapted from Zhuang's work (Zhuang 1999). The CD agent is not considered a part of the C^2I agent. The C^2I agent and CD agent run independently in the experiments.
- Design information database: In Plane World, a global database is used to store the public design-related information such as relations, requirements, and present and past values of variables for conflict detection and Conflict Cause Identification.

In the client-server structure, detailed function-implementation is encapsulated within the objects (i.e. the server and clients). The objects communicate via interface-function calls and messages. Therefore, different objects can be developed concurrently by different development teams.

Also, the client-server structure enhances the maintainability and future growth of the program. In Plane World, the detailed implementation of database communication functions is encapsulated in the global server and masked from the client agents. Therefore, in developing the client agents, the programmer only needs to focus on his own part of work and the interface provided by the server rather than being concerned with the detailed implementation within the server program, which greatly reduces the workload.

In the next chapter, the Plane World design experiments and the results will be discussed.

CHAPTER 6
EXPERIMENTS AND RESULTS

In this section, experiments involving example designs are described and the results are discussed.

6.1 How Were the Experiments Conducted?

In order to demonstrate and test the Conflict Cause Identification (C^2I) methods discussed in previous chapters, several experiments were conducted. The purposes of the experiments are testing if the C^2I software agent works in the web-based design environment and partially validating the overall C^2I methods developed in this study, including the Agent Behavior Model (ABM), the derived Design Process Graph (DPG), and the inverse mappings from VIP/requirement type to the possible conflict causes. In these experiments both human design agents and the software C^2I agent were involved, with conflict detection (CD) results from the CD agent modified from Zhuang's work (Zhuang 1999).

As Figure 6-1 illustrates, the investigator created a potential conflict situation (i.e. a combination of conflict cause and requirement type) in a collaborative example design of a 2-D plane. Then design agents sought to coordinate their design in the presence of these various conflict situations, seeking to satisfy the design requirements. As a result, explicit symptoms of conflicts occurred in the design. The design process was constantly monitored by the software CD agent and C^2I agent. Both CD agent and C^2I agent have access to values of variables associated with global, interface-compatibility, and functional-influenced requirements. When a conflict was detected by the CD agent, the C^2I agent referenced the collected relevant design information (i.e. history values of variables), calculated the VIPs, got the requirement types, identified the conflict class and cause (or hard problem), and notified the design agents. Thus, in these experiments, the design agents received real-time feedback from the C^2I agent during their work. The results of conflict cause identified by the C^2I agent were then evaluated and compared (manually by the investigator) with the conflict situation initially created to assess the performance of the C^2I agent.

6.2 Experimental Results

In this section, four Plane World design experiments involving Conflict Cause Identification (C^2I) operation in different conflict situations are presented and discussed. The overall design task was to satisfy all design requirements (listed in Table 5-4).

In some experiments, multiple conflicts involving different requirement types and conflict causes also emerged. They were detected by the CD agent and their causes were identified by the C^2I agent. To avoid confusion, only the ones with one focus of conflict causes are presented here.

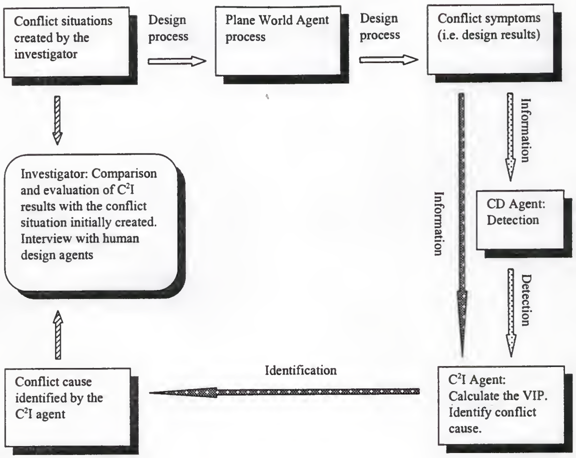

Conflict situations created by the investigator
Design process
Plane World Agent process
Design process
Conflict symptoms (i.e. design results)
Information
Information
CD Agent: Detection
Detection
Investigator: Comparison and evaluation of C²I results with the conflict situation initially created. Interview with human design agents
Identification
Conflict cause identified by the C²I agent
C²I Agent: Calculate the VIP. Identify conflict cause.

Figure 6-1: Experiment design process in Plane World experiments

Experiment 1: Illustrating Different Perception/Conflicting Local Goals/Hard Problem Cause, Interface-Compatibility Requirement

Description of the Experiment

In this experiment, two independent design agents[1] were responsible for the propulsion and wing-structure modules, which were bolted together along the interface as

[1] In this experiment, two graduate students Weijian Wang and Dazhi Yu served as human design agents responsible for the propulsion and wing-structure modules. Tianhong Jiang was responsible for the rest modules.

Figure 6-2 illustrates. The bolts are to be in line and required strong enough to hold the propulsion modules and the wing-structure modules together.

There are also the interface-compatibility requirements that designs of the spacing between bolt/nut holes proposed in the two modules (i.e. the wing-structure and propulsion modules) should be compatible. That is, the difference between the bolt/nut hole spacing in the two modules should be within a small range (here chosen as 0.125 inches):

$$DLWPBNHSpacing < 0.125\, inches \tag{6-1}$$

$$DRWPBNHSpacing < 0.125\, inches \tag{6-2}$$

Where DLWPBNHSpacing is the difference of the bolt/nut hole spacing between the left wing-structure and left propulsion modules and DRWPBNHSpacing is the difference of the bolt/nut hole spacing between the right wing-structure and right propulsion modules.

Experiment Result

Figure 6-3 shows the history values of one of the requirement-related variables (i.e. DLWPBNHSpacing). These values could be viewed by design agents from a window displayed on request.

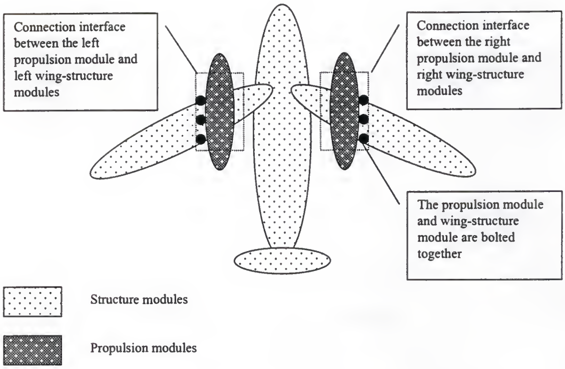
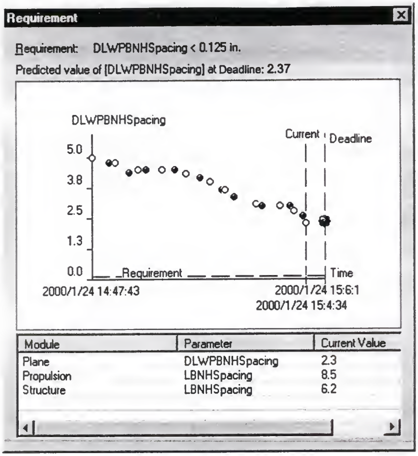

Connection interface between the left propulsion module and left wing-structure modules
Connection interface between the right propulsion module and right wing-structure modules
The propulsion module and wing-structure module are bolted together
Structure modules
Propulsion modules

Figure 6-2: Propulsion and wing-structure modules in the 2-D plane

As Figure 6-3 illustrates, the value of the requirement-related variable (i.e. DLWPBNHSpacing) was determined by choices of the two agents responsible for the propulsion and wing-structure modules. The value of DLWPBNHSpacing was moving toward the acceptable value (i.e. 0.125 inches) but unlikely be there at the deadline. Based on its history, its predicted value (2.37 inches) which was very close to its current value (2.3 inches) was greater than the required value (0.125 inches), indicating a probable conflict in the design process.



Requirement: DLWPBNHSpacing < 0.125 in.

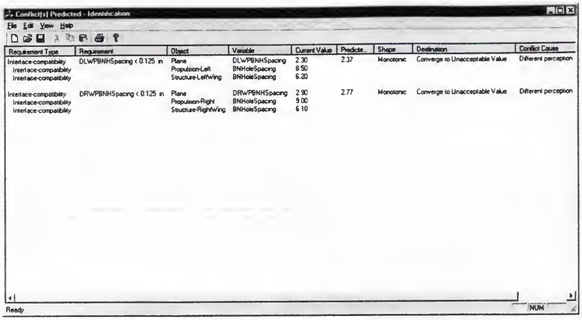
Predicted value of [DLWPBNHSpacing] at Deadline: 2.37

Module	Parameter	Current Value
Plane	DLWPBNHSpacing	2.3
Propulsion	LBNHSpacing	8.5
Structure	LBNHSpacing	6.2

●	Values of the requirement-related variable resulting from a design change in the propulsion agent's module (i.e. left propulsion module)
○	Values of the requirement-related variable resulting from a design change in the structure agent's module (i.e. left wing module)
✸	Predicted value of the requirement-related variable at deadline

Figure 6-3: History values of the requirement-related variable DLWPBNHSpacing

This conflict was detected by the CD agent and identified by the C^2I agent, which sent a notification (window, Figure 6-4) to all design agents in the Plane World experiment.



Requirement Type	Requirement	Object	Variable	Current Value	Predicte...	Shape	Destination	Conflict Cause
Interface-compatibility	DLWPBNHSpacing < 0.125 in	Plane	DLWPBNHSpacing	2.30	2.37	Monotonic	Converge to Unacceptable Value	Different perception
Interface-compatibility		Propulsion-Left	BNHoleSpacing	8.50				
Interface-compatibility		Structure-LeftWing	BNHoleSpacing	6.20				
Interface-compatibility	DRWPBNHSpacing < 0.125 in	Plane	DRWPBNHSpacing	2.90	2.77	Monotonic	Converge to Unacceptable Value	Different perception
Interface-compatibility		Propulsion-Right	BNHoleSpacing	9.00				
Interface-compatibility		Structure-RightWing	BNHoleSpacing	6.10				

Figure 6-4: Conflict(s) Predicted window of Experiment 1

In the C^{2}I window illustrated in Figure 6-4, there are nine columns as follows:

- Requirement type: the type of requirement involved in the conflict. The requirement type was determined and input into the database along with the requirement at the beginning of the experiment. The C^{2}I agent retrieved the requirement type from the database in the inverse identification process. In this experiment, "Interface-compatibility" indicated the requirement was of interface-incompatibility type.
- Requirement: the requirement that is not being satisfied in the design and indicates a possible conflict. In this experiment, there was an interface-compatibility requirement that the difference of bolt/nut-hole spacing between the propulsions and wing structure (i.e. DLWPBNHSpacing for left

wing/propulsion and DRWPBNHSpacing for right wing/propulsion) should be less than 0.125 inches.

- Object: the objects involved in the conflicts. In this experiment, the interface-compatibility requirement was imposed on the Plane level, involving two types of modules, i.e. the left and right propulsion modules (i.e. Propulsion-Left and Propulsion-Right) and the left and right wing structure modules (i.e. Structure-LeftWing and Structure-RightWing).
- Variable: the variables involved in the conflicts, including the requirement-related variables and module-variables whose values contribute to the values of the requirement-related variables.

  In this experiment, the requirement (Equation 6-1) involved the requirement-related variable DLWPBNHSpacing and module-variables BNHoleSpacing (i.e. the spacing between bolt/nut-holes) in the left wing structure and left propulsion modules whose values determined the value of DLWPBNHSpacing. Similarly, requirement (Equation 6-2) involved the requirement-related variable DRWPBNHSpacing and module-variables BNHoleSpacing (i.e. the spacing between bolt/nut-holes) in the right wing structure and right propulsion modules whose values determined the value of DRWPBNHSpacing.
- Current value: the current value of each variable.
- Predicted value: the predicted value of the requirement-related variable at the deadline time.

- Shape: path shape of the requirement-related variable values. For each requirement-related variable, it describes how the variable evolves in the design process. In this experiment, the path shape of both DLWPBNHSpacing and DRWPBNHSpacing were monotonic.
- Destination: predicted path destination of the values of the requirement-related variables. In this experiment, the destinations of both DLWPBNHSpacing and DRWPBNHSpacing were converging to unacceptable values.
- Conflict cause: the fundamental cause of the conflict. Based on the VIPs and the requirement type, the C^2I agent identified this conflict as due to one of the three causes: different perception, conflicting local goals, or hard problem. The present C^2I method cannot distinguish them from each other. Due to the limited width of the monitor-screen, only the different perception cause was displayed in Figure 6-5. The other two could be viewed by clicking the extending cursor on the right bottom of the window.

Interview Results and Evaluation

From the interview, conducted after the experiment, with design agents involved in this experiment, the following facts were obtained:

- The design agents were strongly local-goal and local perception oriented. While they were concerned with the global goals and willing to cooperate, the design agents made their module designs primarily based on their own perceptions. For instance, the propulsion agents preferred bigger and few bolts with larger spacing between the holes that would save space for other connections at the interface (i.e. the electric, fuel and fluid) as illustrated in Figure 6-5. He chose to start at 10 inches for the spacing between the holes. However, the structure agent said his primary concern was to avoid stress concentration and to distribute the load more evenly. He preferred more but smaller bolts with smaller spacing between the holes as Figure 6-5 illustrates. He chose to start at 5 inches.

- The design agents were reluctant to adjust their designs at the interface to reach a compromise when the adjustment would cause changes of the internal designs and result in more cost and slower internal design progress.

The results of the interviews indicated that the design agents behaved generally as predicted by the ABM rules in this work, thus indicating reasonable validity of these rules.

<u>Experiment 2: Illustrating Conflicting Local Goals Cause, Global Requirement</u>

<u>Description of the Experiment</u>

This experiment emphasized on a global requirement imposed on the overall plane weight, which involves four types of modules, i.e. cargo, fuel tank, propulsion, and structure, designed by different design agents[2]. Equations 6-3 and 6-4 show the requirement and the involved module variables.

[2] In this example, three graduate students Weijian Wang, Dazhi Yu, and Yungfei Feng served as human design agents responsible for the fueltank, propulsion, and structure modules of the plane. Tianhong Jiang was responsible for the other modules.

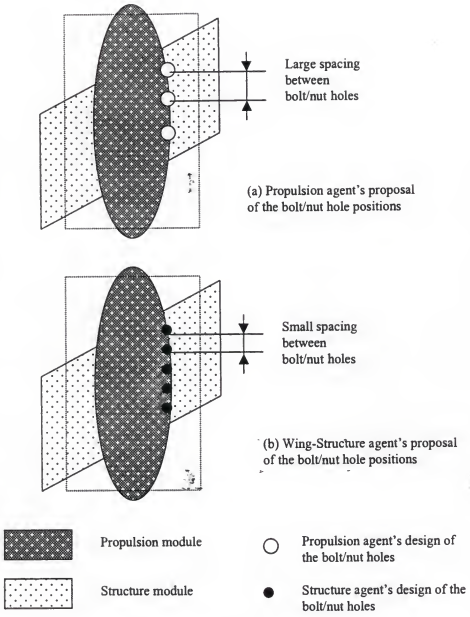

Large spacing between bolt/nut holes
(a) Propulsion agent's proposal of the bolt/nut hole positions
Small spacing between bolt/nut holes
(b) Wing-Structure agent's proposal of the bolt/nut hole positions
Propulsion module
Structure module
Propulsion agent's design of the bolt/nut holes
Structure agent's design of the bolt/nut holes

Figure 6-5: Different preferred designs of the bolt/nut-holes (i.e. spacing) on interface between the propulsion and wing-structure modules

$$Plane.Weight < 25{,}000\,lb. \qquad (6\text{-}3)$$

$$Plane.Weight = \begin{pmatrix} Cargo.Weight + \\ FuelTank.Weight + \\ Propulsion.Weight + \\ Structure.Weight \end{pmatrix} \qquad (6\text{-}4)$$

$$Plane.Range > 2000\,miles \qquad (6\text{-}5)$$

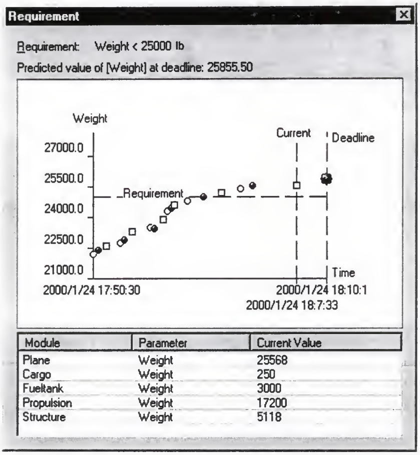
In the design, the agents also sought to cooperate in meeting the global requirement on the Plane's range (Equation 6-5). Meanwhile, each agent has his own local goals and other global goals. Some of the local goals of these design agents were connected to the weight of their modules via functions (Table 6-1).

Comparable relations that apply to range are shown in Table 5-3, Section 5.2.1.

Table 6-1: Local goals and related module weight

MODULE AGENTS	LOCAL GOAL VARIABLE	FUNCTIONAL RELATIONS TO MODULE WEIGHT
Cargo	CargoWeight	$Cargo.Weight = \dfrac{Cargo.WeightCoeff}{Cargo.CargoWeightCoeff} \times Cargo.CargoWeight$
FuelTank	Fuel	$FuelTank.Weight = \dfrac{FuelTank.WeightCoeff}{FuelTank.FuelCoeff} \times FuelTank.Fuel$
Propulsion	Thrust	$Propulsion.Weight = \dfrac{Propulsion.Thrust}{Propulsion.ThrustWeightRatio}$
Structure	Area	$Structure.Weight = Structure.WeightCoeff \times Stucture.Area$

Experiment Result



Requirement: Weight < 25000 lb

Predicted value of [Weight] at deadline: 25855.50

Module	Parameter	Current Value
Plane	Weight	25568
Cargo	Weight	250
Fueltank	Weight	3000
Propulsion	Weight	17200
Structure	Weight	5118

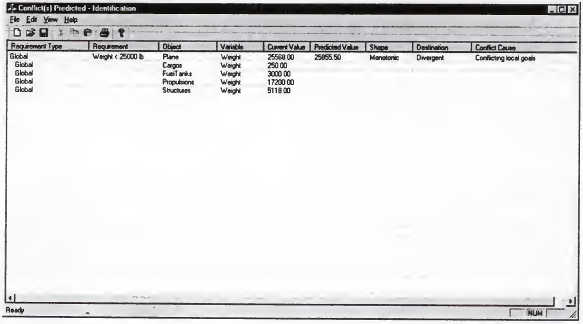
○	Values of the requirement-related variable resulting from a design change in the fueltank agent's modules (i.e. Fueltank modules)
●	Values of the requirement-related variable resulting from a design change in the propulsion agent's modules (i.e. Propulsion modules)
□	Values of the requirement-related variable resulted from a design change in the structure agent's modules (i.e. Structure modules)
✹	Predicted value of the requirement-related variable at deadline

Figure 6-6: History values of the requirement-related variable Weight of the plane

The history values of the requirement-related variables (i.e. the plane weight) are shown in Figure 6-6. These could be viewed by design agents from a window upon request.

As Figure 6-6 illustrated, the value of the requirement-related variable (i.e. weight of the plane) was determined by values of weights of four types of modules (i.e. the cargo, fueltank, propulsion, and structure modules). In pursuing the requirement imposed on the plane range, the value of the plane weight increased steadily beyond the required value, indicating a probable conflict in the design.

Note that in this situation, cargo modules influenced the weight but were not directly involved in the probable conflict because they did not participate in the feedback loop illustrated in Figure 6-8.



Requirement Type	Requirement	Object	Variable	Current Value	Predicted Value	Shape	Destination	Conflict Cause
Global	Weight < 25000 lb	Plane	Weight	25568 00	25855.50	Monotonic	Divergent	Conflicting local goals
Global		Cargos	Weight	250 00				
Global		FuelTanks	Weight	3000 00				
Global		Propulsions	Weight	17200 00				
Global		Structures	Weight	5118 00				

Figure 6-7: Conflict(s) Predicted window of Experiment 2

This conflict was detected by the CD agent and identified by the C^2I agent. The C^2I agent sent each design agent a notification ("Conflict(s) Predicted" window, Figure 6-7).

The window (Figure 6-7) interpretation is similar to that given in Experiment 1. Here, we focus on the following information provided by the C^2I agent:

- Requirement type: The requirement involved in this conflict was a global requirement.
- Object: Here, the plurals in object name indicated all modules of the same types. For instance, "Structures" represented all Structure modules. It should be pointed out that the values of cargo modules' weight were also contributed to the plane weight value even though they were not ohterwise involved in the conflict.
- Shape: In this experiment, the path shape of history values of the global plane weight was identified as monotonic and steadily increasing.
- Destination: In this experiment, the destination of the plane weight was identified as divergent, i.e. the variable value increase steadily beyond the required value.
- Conflict cause: based on the VIPs and the requirement type, the C^2I agent identified the conflict cause as "conflict local goals" in this experiment.

Interview Results and Evaluation

From the post-experiment interview with design agents involved in this experiment, we have found the following facts[3]:

- The design agents were honestly cooperative in design, seeking to meet the global design requirement such as the requirement on the plane's range.
- In pursuing the global requirement (i.e. Range), it appeared the design agents made their module designs primarily based on their own local goals. The agents seemed uninterested in the non-functional goal of weight and gave it little attention. For instance, the fueltank agent considered his local goal (i.e. more fuel) as most important in achieving the global goal (i.e. Range). The propulsion agent argued that more powerful propulsion modules would fly the plane faster and increase its range. To him, the major effort to achieve the range requirement was to build bigger propulsion modules with more thrust, which is his local goal. Meanwhile, the structure agent claimed his role was as important as those of the other agents were. To him, the local goal was to increase the size of the plane structure (i.e. Area in the 2-D plane) for more internal space to accommodate the bigger propulsion and fueltank modules.
- It appeared the design agents were reluctant to change their local goals in design. When one agent did some change to his module, the associated agents

[3] Note that this is a highly simplified plane design model. The real world airplane design is much more complicated. For instance, the approach to achieve the range requirement is also concerned with technology available (i.e. more efficient propulsion unit and lighter composite structure) and the budget.

tended to do some local adjustment seeking to accommodate the former one's design change if the adjustment did not hurt their own local performance. However, the design agents were reluctant to make a compromise design that satisfied the global goal but required them to reduce their local performance. Thus, the design fell into a positive feedback loop and steadily increased the plane weight as Figure 6-8 illustrated.

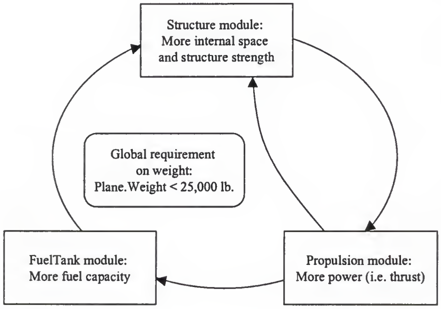


Figure 6-8: Positive feedback loop in the Plane World design experiment

The interview with the design agents again suggested that they did in general behave as predicted by the ABM rules developed in this work.

Experiment3: Illustrating Conflicting Local Goals/Different Perception, Functional-Influenced Requirement

Description of the Experiment

In this experiment, three design agents[4] were involved. Here focus was on the interactions between propulsion and tail-structure modules and between propulsion and control modules in the Plane World design.

In the former interaction, the exhaust of propulsion modules[5] would cause severe heat and vibration loading on the tail-structure modules if the propulsion modules are not properly located, resulting in severe damage to the structure. In other words, the heat and vibration loading on the tail-structure modules is influenced by the location of the propulsion modules via functions. Thus, there is a set of functional-influenced requirements for the tail-structure to be cleared from the propulsion modules' exhaust, that is, the distance from the outer-tip of the tails to the propulsion's exhaust effect zone (i.e. DXLeftProp and DXRightProp) should be greater than zero, as Equations 6-6 and 6-7 show.

$$Structure_{Tail}.DXLeftProp > 0 \tag{6-6}$$

$$Structure_{Tail}.DXRightProp > 0 \tag{6-7}$$

The equations to calculate values of DXLeftProp and DXRightProp were as follows:

[4] In this example, three graduate students, Weijian Wang, Dazhi Yu, and Yungfei Feng served as human design agents responsible for the propulsion, structure, and control modules of the plane. Tianhong Jiang was responsible for the other modules.

[5] Here, it was assumed that the plane used jet propulsion and the exhaust was always oriented aft ward.

$$Structure_{Tail}.DXLeftProp = \begin{matrix}(Structure_{Tail}.X - Propulsion_{Left}.X) - \\ (Structure_{Tail}.Width + Propulsion_{Left}.Width) \times 0.5\end{matrix} \quad (6\text{-}8)$$

$$Structure_{Tail}.DXRightProp = \begin{matrix}(Propulsion_{Right}.X - Structure_{Tail}.X) - \\ (Structure_{Tail}.Width + Propulsion_{Right}.Width) \times 0.5\end{matrix} \quad (6\text{-}9)$$

Where X is the coordinate of objects (i.e. structure and propulsion modules) in x-axis. Width is the width of these objects.

The location of the propulsion units is specified by the propulsion agent. To achieve his local goal (i.e. less loading, adequate structure safety factor), the tail-structure agent is expected to encourage the propulsion agent to put the propulsion modules an adequate distance away from the fuselage (Figure 6-9a).

On the other hand, there is an interaction between the propulsion and control modules. The yaw controllability of the plane is influence by the location of the propulsion modules, especially when one of the propulsion modules is shut down. Thus, there is a set of functional-influenced requirements that the left and right propulsion modules should be close to the fuselage within a certain range (Equation 6-10 and 6-11).

$$Control.DXFuselageLProp < 38\ ft. \quad (6\text{-}10)$$

$$Control.DXFuselageRProp < 38\ ft. \quad (6\text{-}11)$$

The equations to calculate values of DXFuselageLProp and DXFuselageRProp are:

$$Control.DXFuselageLProp = Structure_{Fuselage}.X - Propulsion_{Left}.X \quad (6\text{-}12)$$

$$Control.DXFuselageRProp = Propulsion_{Right}.X - Structure_{Fuselage}.X \quad (6\text{-}13)$$

One of the local goals of the control agent is to seek good yaw controllability of the plane. To achieve the local goal, the control agent is expected to encourage the propulsion agent to put the propulsion modules close to the fuselage (Figure 6-9b).

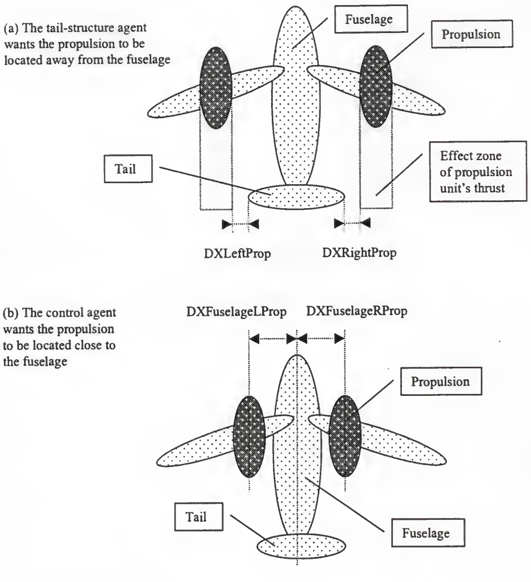
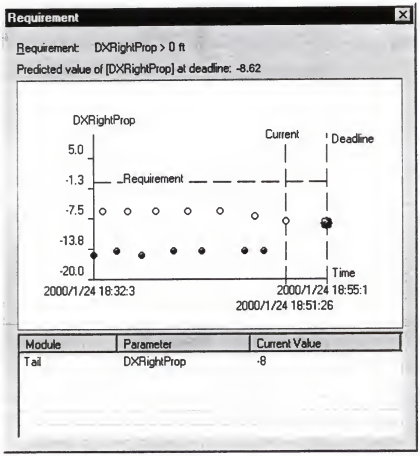

(a) The tail-structure agent wants the propulsion to be located away from the fuselage
Fuselage
Propulsion
Tail
Effect zone of propulsion unit's thrust
DXLeftProp
DXRightProp
(b) The control agent wants the propulsion to be located close to the fuselage
DXFuselageLProp
DXFuselageRProp
Propulsion
Tail
Fuselage

Figure 6-9: An example of conflicting local goals/different perception between the tail-structure and control agents regarding the propulsion modules' locations

Experiment Result

The history values of one of the requirement-related variables (i.e. DXRightProp) are displayed in Figure 6-10. These can be viewed by design agents as desired.



Module	Parameter	Current Value
Tail	DXRightProp	-8

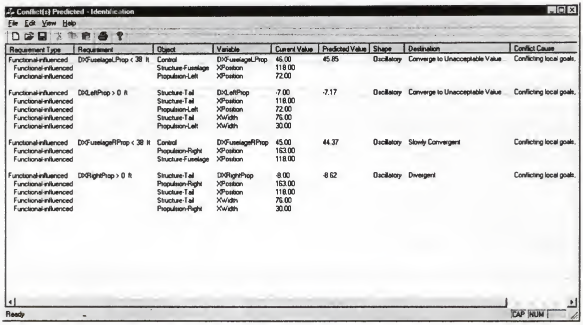
○	Values of the requirement-related variable (i.e. DXRightProp) in tail-structure module resulting from the change of propulsion module's location on the structure agent's request
●	Values of the requirement-related variable (i.e. DXRightProp) in tail-structure module resulting from the change of propulsion module's location on the control agent's request
✹	Predicted value of the requirement-related variable at deadline

Figure 6-10: History values of the requirement-related variable DXRightProp

Similarly, the plot of history values of other requirement-related variables (DXLeftProp, DXFuselageLProp, and DXFuselageRProp) could also be viewed by design agents on request.

Figure 6-10 illustrates the value of the functional-influenced requirement-related variable (DXRightProp) as influenced by the right propulsion module's position. It oscillated in the design due to different requests from the design agents responsible for the control and tail-structure modules. The predicted value of DXRightProp was not converging toward its required value (a value greater than zero), indicating a probable conflict in the design.

This conflict was detected by the CD agent and identified by the C^2I agent. The C^2I agent sent a notification (window, Figure 6-11) to the design agents.



Requirement Type	Requirement	Object	Variable	Current Value	Predicted Value	Shape	Destination	Conflict Cause
Functional-influenced	DXFuselageLProp < 38 ft	Control	DXFuselageLProp	46.00	45.85	Oscillatory	Converge to Unacceptable Value	Conflicting local goals.
Functional-influenced		Structure-Fuselage	XPosition	118.00				
Functional-influenced		Propulsion-Left	XPosition	72.00				
Functional-influenced	DXLeftProp > 0 ft	Structure-Tail	DXLeftProp	-7.00	-7.17	Oscillatory	Converge to Unacceptable Value	Conflicting local goals.
Functional-influenced		Structure-Tail	XPosition	118.00				
Functional-influenced		Propulsion-Left	XPosition	72.00				
Functional-influenced		Structure-Tail	XWidth	76.00				
Functional-influenced		Propulsion-Left	XWidth	30.00				
Functional-influenced	DXFuselageRProp < 38 ft	Control	DXFuselageRProp	45.00	44.37	Oscillatory	Slowly Convergent	Conflicting local goals.
Functional-influenced		Propulsion-Right	XPosition	163.00				
Functional-influenced		Structure-Fuselage	XPosition	118.00				
Functional-influenced	DXRightProp > 0 ft	Structure-Tail	DXRightProp	-8.00	-8.62	Oscillatory	Divergent	Conflicting local goals.
Functional-influenced		Propulsion-Right	XPosition	163.00				
Functional-influenced		Structure-Tail	XPosition	118.00				
Functional-influenced		Structure-Tail	XWidth	76.00				
Functional-influenced		Propulsion-Right	XWidth	30.00				

Figure 6-11: Conflict(s) Predicted window in Experiment 3

Here the Conflict(s) Predicted window (Figure 6-11) is again interpreted similar to that given in Experiment 1. Specifically, the window provided design agents the following information:

- Requirement type: in this experiment, the requirements involved in the conflicts were functional-influenced requirements.
- Shape: In this experiment, the path shapes of the requirement-related variables (i.e. DXLeftProp, DXFuselageLProp, DXRightProp, and DXFuselageRProp) were oscillatory.
- Destination: In this experiment, the destination of DXRightProp was divergent. The destination of DXFuselageRProp was slowly convergent. The destinations of DXLeftProp and DXFuselageLProp were converging to an unacceptable value/stable at several values.
- Conflict cause: Based on the VIPs and the requirement type, the C^2I agent identified the conflict cause as either conflicting local goals or different perception. Currently, these two possible causes cannot be distinguished from each other from the VIPs. Due to the limited monitor-screen size, only one cause (conflicting local goals) was displayed. The other one (different perception) could be viewed by clicking the extension cursor at the right bottom of the window.

<u>Interview and Evaluation</u>

From the interview with design agents following this experiment, we found the following facts:

- In the design, the agents were strongly local goal-oriented. To meet the functional-influenced requirement, each agent expected other agents to be cooperative in achieving his local goal. For instance, the control agent sent request messages to the propulsion agent asking him to put the propulsion module close to the fuselage module for better yaw-controllability. Meanwhile, the tail-structure agent sent opposite requests to the propulsion agent asking him to move the propulsion modules away from the fuselage so that the tail module could be cleared from the exhaust of the propulsion modules.
- When the design agents' local goals were conflicting with each other, the design agents were reluctant to change their local goals dramatically. A reason stated by the agents was that changing local goals often involved complex adjustment of existent internal module designs.
- In the design, the propulsion agent was cooperative, trying to satisfy the requests from the other two agents since his major local goals (i.e. thrust) were not much affected by the propulsion modules' location as he stated.

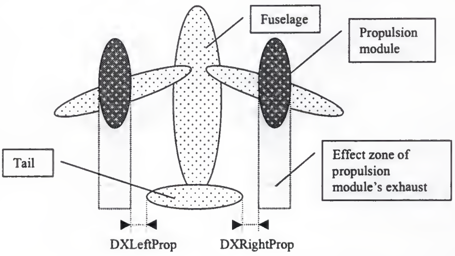
The interview with the design agents suggested that they behaved in general as predicted by the ABM rules developed in this work.

Experiment 4: Illustrating Inadequate Communication Cause, Functional-Influenced Requirement

Description of the Experiment

This experiment was similar to Experiment 3 but with different focus (i.e. communication issue). Here the people involved in the experiment are assigned in agent roles. Also, the control agent is prevented from sending messages to the propulsion agent.

In this experiment, two design agents[6] were responsible for the propulsion and tail-structure modules of the plane. There is an interaction between the propulsion modules and tail-structure modules that the exhaust of propulsion modules might cause over heat and vibration loading on the tail-structure module if they are improperly located (Figure 6-12).



The exhaust of the propulsion modules will cause high loading of heat, noise and vibration on the tail if the tail is within the propulsion module's exhaust effect zone, i.e. (1) DXLeftProp < 0, or (2) DXRightProp < 0.

Figure 6-12: Interaction between the propulsion modules and tail-structure modules in the design process

[6] In this example, two graduate students, Yunfei Feng and Dazhi Yu served as human design agents responsible for the propulsion and structure modules of the plane.

To protect the tail structure, it is required to keep the tail module out of the effect zone of the propulsion modules' exhaust. In other word, the distance from the outer tip of the tail to the effect zone of the propulsion module's exhaust (i.e. DXLeftProp and DXRightProp in the tail-structure module) should be greater than zero:

$$Structure_{Tail}.DXLeftProp > 0 \tag{6-14}$$

$$Structure_{Tail}.DXRightProp > 0 \tag{6-15}$$

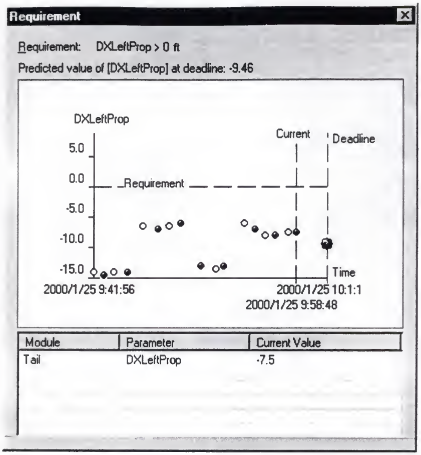
Equation 6-8 and 6-9 show that values of DXLeftProp and DXRightProp were determined by values of the locations and width of both propulsion and tail-structure modules.

Experiment Result

The history values of one of the requirement-related variables (i.e. DXLeftProp) are displayed in Figure 6-13. These can be viewed by design agents on request.

As Figure 6-13 illustrated, the values of the requirement-related variable (i.e. DXLeftProp) were determined by values of locations and width of the propulsion modules and the tail-structure modules. These requirement-related variable values were divided into several segments in the design process in this experiment. Within each segment, the values changed marginally. However, the values between two adjacent segments changed significantly. The predicted value was diverging away from the required value, indicating a probable conflict.

This conflict was detected by the CD agent and identified by the C^2I agent. The C^2I agent sent a notification (window, Figure 6-14) to the design agents.


Requirement

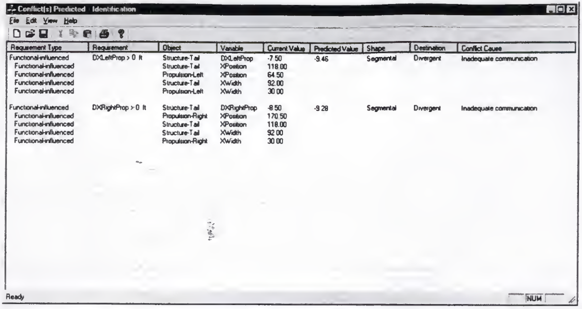
Requirement: DXLeftProp > 0 ft

Predicted value of [DXLeftProp] at deadline: -9.46

Module	Parameter	Current Value
Tail	DXLeftProp	-7.5

○	Values of the requirement-related variable (i.e. DXLeftProp) in tail-structure module resulting from the change of propulsion module's design
●	Values of the requirement-related variable (i.e. DXLeftProp) in tail-structure module resulting from the change of structure module's design
⬢	Predicted value of the requirement-related variable at deadline

Figure 6-13: History values of the requirement-related variable DXLeftProp



Requirement Type	Requirement	Object	Variable	Current Value	Predicted Value	Shape	Destination	Conflict Cause
Functional-influenced	DXLeftProp > 0 ft	Structure-Tail	DXLeftProp	-7.50	-9.46	Segmental	Divergent	Inadequate communication
Functional-influenced		Structure-Tail	XPosition	118.00				
Functional-influenced		Propulsion-Left	XPosition	64.50				
Functional-influenced		Structure-Tail	XWidth	92.00				
Functional-influenced		Propulsion-Left	XWidth	30.00				
Functional-influenced	DXRightProp > 0 ft	Structure-Tail	DXRightProp	-8.50	-9.28	Segmental	Divergent	Inadequate communication
Functional-influenced		Propulsion-Right	XPosition	170.50				
Functional-influenced		Structure-Tail	XPosition	118.00				
Functional-influenced		Structure-Tail	XWidth	92.00				
Functional-influenced		Propulsion-Right	XWidth	30.00				

Figure 6-14: Conflict(s) Predicted window in Experiment 4

Again, the Conflict(s) Predicted window interpretation is similar to that given in Example 1. Here, the window provided design agents the following information:

- Requirement type: In this experiment, the requirements involved in the conflict were functional-influenced requirements.
- Shape: Here, the path shapes of both requirement-related variables (i.e. DXLeftProp and DXRightProp) were segmental.
- Destination: The destinations of both requirement-related variables (i.e. DXLeftProp and DXRightProp) were divergent in this experiment.
- Conflict cause: based on the VIPs and the requirement type in this experiment, the C^{2}I agent identified the conflict cause as inadequate communication.

<u>Interview and Evaluation</u>

From the post-experiment interview with design agents, the following information was obtained:

- In the design, design agents were strongly local goal oriented. For instance, the structure agent was concerned with the integrity of his structure module and the threat of the exhaust of the propulsion module. Meanwhile, the propulsion agent focused on his local goals (i.e. thrust of the propulsion modules etc.).
- While the design agent (i.e. the propulsion agent) sought to be cooperative in design, it appeared he was primarily concerned with his own local goals and tended to ignore the requests from other agents (i.e. the structure agent) when the requests had little to do with his local goals. The propulsion agent claimed he was willing to cooperate in design but he preferred considering the structure agent's requests after he had made substantial effort in his module design. As a result, the delay was not long in some cases, but it was excessively long in other cases. The delay caused the agents to be unaware of other agents' design changes and to use obsolete information in their designs, for instance, the structure agent enlarged the tail-structure module for better performance in the experiment. However, the propulsion agent was not aware of it and did not adjust the propulsion modules' locations for a long period of time, which resulted in conflict in the design.

Again, the result of the interview with the design agents indicated agent behavior consistent with the ABM rules developed in this work.

<u>Additional Experiments</u>

In addition to the experiments discussed above, six more experiments were conducted by the same group of design agents. The results are listed Table 6-2.

Table 6-2: Results of additional experiments

EXPERIMENT INDEX	REQUIREMENT INVOLVED IN THE CONFLICT	CONFLICT IDENTIFICATION RESULT
5	Interface-compatibility requirement imposed on the bolt/nut hole spacing at the interface between the wing-structure and propulsion modules: $DLWPBNHSpacing < 0.125\ inches$ $DRWPBNHSpacing < 0.125\ inches$	Successful in identification. VIPs of DLWPBNHSpacing and DRWPBNHSpacing: Monotonic, slowly convergent, Conflict cause identified: Different perception.
6	Global requirement imposed on the plane weight and range: $Plane.Weight < 25{,}000\ lb.$ $Plane.Range > 2000\ miles$	Successful in identification. VIP_1 of Plane.Weight: Monotonic, divergent, VIP_2 of Plane.Range: Monotonic, converge to an unacceptable value. Conflict cause identified based on VIP_1: Conflicting local goals. For VIP_2: it is identified as Hard problem.
7	Functional-influenced requirements imposed on the clearance distance of the tail to the effect zone of the propulsion module exhaust: $Structure_{Tail}.DXLeftProp > 0$ $Structure_{Tail}.DXRightProp > 0$	Successful identification. VIPs of DXLeftProp and DXRightProp: Segmental, slowly convergent. Conflict cause identified: Inadequate communication.

8	Functional-influenced requirements imposed on the clearance distance of the tail to the effect zone of the propulsion module exhaust: $Structure_{Tail}.DXLeftProp > 0$ $Structure_{Tail}.DXRightProp > 0$	Partial success in identification. VIP$_1$ of DXLeftProp: Segmental, divergent, VIP$_2$ of DXRightProp: Chaotic, divergent. Conflict cause identified based on VIP$_1$: inadequate communication, Conflict cause identified based on VIP$_2$: no result.
9	Functional-influenced requirements (in the structure modules) imposed on the clearance distance of the tail to the effect zone of the propulsion module exhaust: $Structure_{Tail}.DXLeftProp > 0$ $Structure_{Tail}.DXRightProp > 0$ Functional-influenced requirements (in the control module) imposed on the distance between the propulsion modules and the fuselage-structure modules: $Control.DXFuselageLProp < 38\ ft.$ $Control.DXFuselageRProp < 38\ ft.$	Success in identification. VIPs of DXLeftProp and DXRightProp: Oscillatory, slowly convergent. VIPs of DXFuselageLProp and DXFuselageRightProp: Oscillatory, divergent. Conflict cause identified: Conflicting local goals.
10	Functional-influenced requirements (in the structure modules) imposed on the clearance distance of the tail to the effect zone of the propulsion module exhaust: $Structure_{Tail}.DXLeftProp > 0$ $Structure_{Tail}.DXRightProp > 0$ Functional-influenced requirements (in the control module) imposed on the distance between the propulsion modules and the fuselage-structure modules: $Control.DXFuselageLProp < 38\ ft.$ $Control.DXFuselageRProp < 38\ ft.$	Partial success in identification. VIP$_1$ of DXLeftProp and DXRightProp: Oscillatory, slowly convergent. VIP$_2$ of DXFuselageLProp and DXFuselageRightProp: Chaotic, divergent. Conflict cause identified based on VIP$_1$: Conflicting local goals, Conflict cause identified based on VIP$_2$: no result.

Among the ten experiments conducted, the C^2I agent made successful identification in eight experiments and partially successful identification in the other two experiments. The rate of successful identification is estimated to be roughly 80 percent.

6.3 Summary of Experimental Results

From the experiment results presented and discussed above, it has been determined that the software C^2I agent based on the C^2I methods developed in this project can work effectively in identifying conflict class and cause in a web-based distributed design system (i.e. the Plane World system). The results of VIPs and conflict causes identified by the C^2I agent were generally consistent with the results from the investigator's manual analysis.

In the design process, the design agents in general behaved as predicted by the ABM rules.

In current work, the C^2I agent is designed to identify the conflict causes in category-level[7]. In the future, the C^2I methods can perhaps be improved to be identify the conflict causes in more refined case-level.

[7] Refer to Section 3.4.

CHAPTER 7
DISCUSSION AND CONCLUSIONS

The goal of this dissertation work is to provide a computational model to identify the conflict causes in web-based design systems, thus, supporting conflict resolution in distributed concurrent engineering (CE) designs. This work focuses on the autonomous identification of conflict class and cause in the distributed collaborative design, using human design agents.

In this study, a model of Conflict Cause Identification in a web-based design system (simply called the C^2I model) has been proposed. The C^2I model is generic and can provide support in a wide variety of domains. It includes a partially validated model of how human design agents behave in conflict situations in the web-based cooperative design and how the Conflict Cause Identification should operate.

Meanwhile, a model web-based design system has been developed to demonstrate the application of the C^2I model in distributed cooperative design.

In this work, it is principally concluded that:

- A conflict can be described along two basis dimensions: requirement type and fundamental cause that are important in determining its behavior. Conflicts can be grouped into different classes that are combinations of different requirement types and fundamental causes.

- Value Indicator Patterns (VIPs), based on the published variable values, are information consistently available to support the C^2I operation in design processes. The VIPs can be partitioned along two basic dimensions: path shape and destination. The requirement type also plays an important role in identifying the fundamental cause of a conflict.
- Using rules in natural language, the conflict-related Agent Behavior Model (ABM) is suitable in describing how the human design agents behave in conflict situations in cooperative designs.
- Based on the ABM, the Design Process Graph (DPG) can be created to describe how the variables evolve in the design in the presence of different conflict situations.
- As a result, the mappings from different conflict causes/requirement types to associated VIPs have been built. The inverse of these mappings (from the VIPs/requirement types to the possible conflict causes) is used for the inverse identification process.
- A software C^2I agent, that can monitor the ongoing design process, identify the conflict cause, and notify the design agents when a conflict is detected, has been developed. The C^2I agent developed successfully identified most of the conflict causes during designs in the web-based Plane World system that involved multiple independent human design agents. The C^2I agent is expected to be applicable to other web-based design systems and tasks.

From the literature referenced, it is believed this dissertation study represents a significant addition to previous work in the areas of conflict classification and conflict class identification.

GLOSSARY

Area	Area of the object[1]. Unit: ft^2
BNHoleSpacing	Spacing between the bolt/nut holes in the interface between two modules. Unit: in.
CargoWeight	Cargo capacity of the object (i.e. plane and cargo module) in terms of weight. Unit: lb.
CargoWeightCoeff	Cargo weight coefficient of the cargo module. Unit: lb. / ft^2.
Cost	Initial cost of the object. Unit: \$1,000.
CostCoeff	Initial cost coefficient. For the cargo, fueltank, and structure module, the unit is \$1,000 / ft^2. For the propulsion module, the unit is \$1,000 / lb_{Thrust}.
CostThrustCoeff	The ratio coefficient of cost over thrust; amount of cost per unit thrust. Unit: \$1,000 / lb_{Thrust}.
Drag	Drag of the object. Unit: lb.
DragCoeff	Drag coefficient. Unit: lb / ft^2.

[1] Here object represents plane and modules.

DLWPBNHSpacing	Difference between values of the bolt/nut-hole spacing in the left wing-structure and the left propulsion modules. Unit: in.
DLWPBNHSpacing	Difference between values of the bolt/nut-hole spacing in the right wing-structure and the right propulsion modules. Unit: in.
DXFuselageLProp	Distance from the left propulsion and the centerline of fuselage in spanwise direction (x-direction). Unit: ft.
DXFuselageRProp	Distance between the right propulsion and the centerline of fuselage in spanwise direction (x-direction). Unit: ft.
DXLeftProp	Clearance distance from the tail to the effect zone of the left propulsion module exhaust in spanwise direction (i.e. x-direction), i.e. if the value is greater or equal to zero, the tail is cleared from the left propulsion's exhaust. Unit: ft.
DXRightProp	Clearance distance from the tail to the effect zone of the right propulsion module exhaust in spanwise direction (i.e. x-direction), i.e. if the value is greater or equal to zero, the tail is cleared from the left propulsion's exhaust. Unit: ft.
Fuel	Fuel capacity of the fueltank module. Unit: lb.
FuelCoeff	Fuel coefficient; i.e. fuel capacity per area unit. Unit: lb / ft^2.
FuelRate	Fuel rate, i.e. amount of fuel consumed by the propulsion module per hour. Unit: lb_{Fuel} / hr.
Height	Height of the module. Unit: ft.
Range	Range of the plane. Unit: miles.

Speed	Speed of the plane. Unit: mph.
Strength	Strength of a structure module. Unit: lb / ft^2.
Thrust	Thrust of the propulsion module. Unit: lb.
ThrustWeightRatio	The ratio of thrust over weight of a propulsion module. Unit: lb_{Thrust} / lb_{Weight}.
TSFC	Thrust-specific fuel consumption; i.e. amount of fuel consumed by the propulsion module per (thrust) pound per hour. Unit: lb_{Fuel} / (lb_{Thrust} * hr).
Weight	Weight of the object. Unit: lb.
WeightCoeff	Weight coefficient, i.e. weight per unit area. Unit: lb / ft^2.
Width	Width of the object. Unit: ft.
X	Coordinate of the module center in the spanwise direction.
Y	Coordinate of the module center in the longitudinal direction.

LIST OF REFERENCES

Anderson 1989 John Anderson, Jr., *Introduction to Flight*, Third Edition, McGraw-Hill Series in Aeronautical and Aerospace Engineering, 1989, McGraw-Hill Book Company

Balabonov 1996 V. Balabanov, M. Kaufman, D. L. Knill, D. Haim, O. Golovidov, A. A. Giunta, B. Gross-man, W. H. Mason, L. T. Watson, and R. T. Haftka, Dependence of Optimal Structural Weight on Aerodynamic Shape for a High-Speed Civil Transport, *AIAA Paper 96-4046, Proceedings 6th AIAA/NASA/USAF/ISSMO Symposium on Multidisciplinary Analysis and Optimization, Bellevue, WA,* Sept. 4-6, 1996, pp. 599-612

Benzem 1984 M. Benzem, Consistency of Rule-Based Expert System, *Technique Report, Center for Mathematics and Computational Science*, July 1987

Carrico 1989 Michael A. Carrico, John E. Girard, Jennifer P. Jones, *Building Knowledge Systems*, Intertext Publications, McGraw-Hill Book Company, 1989

Cointe 1997 Christophe Cointe, Nada Matta, Myriam Ribiere, Design Propositions Evaluation: Using Viewpoint to Manage Conflicts in CREOPS2, *ISPE/CE, Concurrent Engineering: Research and Applications*, Rochester August 1997

Davis 1982 Randall Davis, Douglas B. Lenat, *Knowledge-based Systems in Artificial Intelligence*, McGraw-Hill International Book Company, 1982

Easterbrook 1993 S. M. Easterbrook, E. E. Beck, J. S. Goodlet, L. Plowman, M. Sharples, C. C. Wood, A Survey of Empirical Studies of Conflict, *CSCW: Cooperation or Conflict*? Springer-Verlag 1993

Fedrizzi 1988 M. Fedrizzi, J. Kacpryzk, S. Zadrozny, An Interactive Multi-User Decision Support System for Consensus Reaching Using Fuzzy Logic with Linguistic Quantifiers, Vol. 4, Amsterdam: Elsevier Science Publishers, 1988, p 313 - 327

Feldman 1985	M. Feldman, A Taxonomy of Inter-Group Conflict Resolution Strategies, presented at *1985 Annual Conference Developing Human Resources*, 1985
Findler 1995	Nicholas V. Findler, Gregory D. Elder, Multiagent Coordination and Cooperation in a Distributed Dynamic Environment with Limited Resources, *Artificial Intelligence in Engineering*, Vol. 9, 1995, p229-238
Finin 1994	T. Finin et al., KQML as an Agent Communication Language, *Proceeding Third International Conference Information and Knowledge Management*, ACM Press, New York, 1994
Fleischer 1997	M. Fleischer, Jeffrey K. Liker, Concurrent Engineering Effectiveness: Integrating Product Development Across Organizations, Hanser Gardner Publications, 1997
Gamma 1995	Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, Grady Booch, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional Computing, 1995
Giunta 1996	A. A. Giunta, V. Balabanov, D. Haim, B. Grossman, W. H. Mason, L. T. Watson, and R. T. Haftka, Wing Design for a High-Speed Civil Transport Using Design of Experiments Methodology, *AIAA Paper 96-4001, 6th AIAA/NASA/USAF/ISSMO Symposium on Multidisciplinary Analysis and Optimization, Bellevue, WA*, Sept. 4-6, 1996.
Harrington 1995	John V. Harrington, Hossein Soltan, Mark Forskitt, Negotiation in a Knowledge-Based Concurrent Engineering Design Environment, *Expert Systems*, May 1995, Vol. 12, No. 2, p 139-147
Harrington 1996	John V. Harrington, Hossein Soltan, Mark Forskitt, Framework for Knowledge Based Support in a Concurrent Engineering Environment, *Knowledge-Based Systems* 9, 1996, p207-215
Hoffman 1997	Dennis R. Hoffman, An Overview of Concurrent Overview, *Proceedings Annual Reliability and Maintainability Symposium*, 1997, p1-6

Hu 1996	Y. Hu, P. Liu, Y. Yan, D. Zheng, C. Ma, J. Bode, S. Ren, A Multiagent System for the Support of Concurrent Engineering, IEEE 1996, p959-964
Hunnicutt 1988	R. P. Hunnicutt, *Firepower: A History of the American Heavy Tank*, Presidio Press, 1988
Klein 1990	Mark Klein, Conflict Resolution in Cooperative Design, *Ph.D. dissertation, Univ. Ill., Urbana-Champaign*, Jan. 1990
Klein 1991	Mark Klein, Supporting Conflict Resolution in Cooperative Design Systems, *IEEE Transactions on Systems, Man. and Cybernetics*, Vol. 21, No. 6, November-December 1991, p1379-1390
Lander 1997a	Susan E. Lander, Issues in Multiagent Design Systems, *IEEE Expert*, March-April 1997, p18-26
Lander 1997b	Susan E. Lander, R. Lesser, Sharing Metainformation to Guide Cooperative Search among Heterogeneous Reusable Agents, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 9, No. 2, March-April 1997, p193-208
Lin 1996	J. Lin, M. S. Fox, T. Bilgic, A Requirement Ontology for Engineering Design, *Concurrent Engineering: Research and Applications*, Vol. 4, No. 3, September 1996, p 279 - 291
Matta 1996	Nada Matta, Conflict Management in Concurrent Engineering: Modeling Guides, *Proceeding in Conflicts in AI Workshop, ECAI*, Budapest, 1996
Macksey 1988	Kenneth Macksey, *Tank versus Tank*, Salem House Publishers, 1988
Mowbray 1995	T. J. Mowbray, R. Zahavi, *The Essential Corba: Systems Integration Using Distributed Objects*, John Wiley & Sons, New York, 1995
Noore 1992	Afzel Noore, Michael Lawson, Electronic Product Design And Manufacture in a Concurrent Engineering Environment, *IEEE Transactions on Consumer Electronics*, 1992
Owen 1985	W. F. Owen, Metaphor Analysis of Cohesiveness in Small Discussion Groups, *Small Group Behavior*, 16(3), p415 – 424

Ozawa 1997	Masanori Ozawa, Yumi Iwasaki, Mark R. Cutkosky, Multi Disciplinary Early Performance Evaluation via Logical Description of Mechanism: DVD Pick Up Head Example, *Proceedings of DETC'98, 1998 ASME Design Engineering Technical Conferences*, September 13-16, 1998, Atlanta, Georgia.
Pena-Mora 1993	Feniosky Pena-Mora, Duvvuru Sriram, Robert Logcher, Design Rationale for Computer-Supported Conflict Mitigation, *Journal of Computing in Civil Engineering*, Vol. 9, No. 1, January 1995, p57-72
Petrie 1995	C. J. Petrie, T. A. Webster, M. R. Cutkosky, Using Pareto Optimality to Coordinate Distributed Agents, *AIEDAM, Special Issue on Conflict Management in Design*, Vol. 9, No. 4, September 1995, p 269 - 282
Pruitt 1981	D.G. Pruitt, *Negotiation Behavior*, Academic Press, 1981.
Ramesh 1994	B. Ramesh, K. Sengupta, Managing Cognitive and Mixed-Motive Conflicts in Concurrent Engineering, *Concurrent Engineering: Research and Applications*, Vol. 2, No. 3, 1994, p 223-236
Rollinger 1996	Claus R. Rollinger, Conflict Resolution and Communication, *ECAI96, 12th European Conference on Artificial Intelligence*, 1996
Sobieski 1996	J. Sobieszczanski-Sobieski, R. T. Haftka, Multidisciplinary Aerospace Design Optimization: Survey of Recent Developments, *34th AIAA Aerospace Sciences Meeting and Exhibit, AIAA Paper No. 96-0711, Reno, Nevada,* January 15-18, 1996
Suwa 1982	M. Suwa, A. C. Scott, E. H. Shortliffe, An Approach to Verifying Completeness and Consistency in a Rule-Based Expert System, *Proceeding AIAA*, 1982
Sycara 1991	K. P. Sycara, Cooperative Negotiation in Concurrent Engineering Design, *Computer Aided Cooperative Product Development, Proceeding of MIT-JSME Workshop*. D. Sriram, R. Logcher, S. Fukuda (eds.), Cambridge, MA 1991
Tanenbaum 1994	Andrew S. Tanenbaum, *Distributed Operating Systems*, Prentice Hall, 1994
Ullman 1997	David G. Ullman, *The Mechanical Design Process (2nd Edition)*, The McGraw-Hill Companies, Inc., 1997

Weiss 1998	Sholom M. Weiss, Nitin Indurkhya, *Predictive Data Mining: A Practical Guide*, Morgan Kaufmann Publishers, 1998.
Wellman 1995	M. P. Wellman, A Computational Market Model for Distributed Configuration Design, *AIEDAM*, Vol. 9, 1995, p 125 - 133
Werkman 1991	Keith J. Werkman, Negotiation as an Aid in Concurrent Engineering, *Issues in Design Manufacture/Integration*, DE-Vol. 39, ASME 1991, p23-30
Wolff 1983	Diane D. Wolff, Michael L. Parsons, Pattern Recognition Approach to Data Interpretation, Plenum Press, 1983
Zhuang 1999	Ruiqiang Zhuang, Conflict Detection in Web Based Concurrent Engineering Design, *Master degree thesis, University of Florida*, 1999

BIOGRAPHICAL SKETCH

Tianhong Jiang was born in Nanjing, P. R. China, on February 24, 1969. Jiang grew up in China. In July 1992, Jiang received his Bachelor of Engineering degree, major engineering mechanics, at Tsinghua University. Following two-year graduate study in aerospace engineering at the Beijing University of Aeronautics and Astronautics, Jiang came to the U.S. to pursue graduate study, major aerospace engineering, at the University of Florida in 1994. In December 1998, Jiang received his Master of Science, major computer science (Concurrent Program), at University of Florida. Currently Jiang is a candidate for the degree of Doctor of Philosophy to be received in May 2000.

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

[illegible]

Gale E. Nevill, Jr., Chairman
Professor of Aerospace Engineering,
Mechanics, and Engineering Science

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

[illegible]

Raphael T. Haftka
Distinguished Professor of Aerospace
Engineering, Mechanics, and
Engineering Science

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

[illegible]

Andrew J. Kurdila
Associate Professor of Aerospace
Engineering, Mechanics, and
Engineering Science

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

[illegible]

Paul Fishwick
Professor of Computer and Information
Science and Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

[illegible]

Joseph Duffy
Graduate Research Professor of
Mechanical Engineering

This dissertation was submitted to the Graduate Faculty of the College of Engineering and to the Graduate School and was accepted as partial fulfillment of the requirements for the degree of Doctor of Philosophy.

May 2000

[illegible]
M. Jack Ohanian
Dean, College of Engineering

Winfred M. Phillips
Dean, Graduate School